£1.00

ELBUG FOR THE ELECTRON

Wee Shuggy

Vol 2 No 1 November 1985

Games

* **Red Alert**
* **Downhill Ski Racer**

PLUS

* **Synchronising Words and Music**
* **Joystick Routine for the Plus 1**
* **Whirlpool Display**

PLUS

* **Latest Add-ons Reviewed**
* **New Games Reviewed**
* **Book Reviews**
* **And much more**

EDITORIAL

ELBUG ONE YEAR OLD

You will find enclosed with this issue of ELBUG a complete index to the entire contents of Volume 1 of the magazine. This has been organised to try and help you if you are seeking information on a particular topic, rather than simply by origianal title. The new ELBUG binders contain provision to allow you to include the index with your set of issues for Volume 1. Now that ELBUG is beginning to mature any comments or ideas for the future will be most welcome.

CHRISTMAS COMES EARLY

Despite my statement in the previous issue this is not the Christmas issue of ELBUG - that will follow next month - but indicates how far ahead each issue has to be planned. This editorial is being written in mid October when much of the work for the December issue (the real Christmas issue) is well under way. This highlights the importance of submitting any contributions well in advance if they are intended for a particular issue or appropriate to a particular time of year.

NEW SOFTWARE FOR THE ELECTRON

You will also see in the section below that our software section have now released a range of our proven software for the Electron in time for the Christmas market. These titles have already proved tremendously popular with BEEBUG members and we are sure that they will be equally attractive to ELBUG readers.

TICE BOARD NOTICE BOARD NOTICE BOARD NOTICE BOA

NEW RELEASES FROM BEEBUGSOFT

Nearly all of BEEBUGSOFT's range of serious software is now available in versions for the Electron. Now that there are several Electron ROM boards on the market, Exmon and Toolkit are also being made available in this format. Exmon is a complete machine code monitor for the Electron, and Toolkit a set of most useful utilities for all Basic programmers. Exmon is also available on cassette.

Other programs on cassette include Masterfile - a much improved Electron version of the well established BBC micro database package - and Superplot - the best thing to happen to graphs since the invention of squared paper.

For the less seriously minded there is Sprite Utilities - a package providing all you need to design and use fast moving, animated, full colour characters in your own Basic games programs - and Paintbox - a sophisticated drawing package that includes outline drawing, shape creation, and fill in many shades and textures.

For beginners there is a Starter Pack with 5 utility programs, 7 games, and a comprehensive manual with hints, tips, and articles explaining all the more difficult aspects of your new Electron.

Full details are contained in the Software Brochure that you receive with this issue of ELBUG. Note that we are now taking Access and Barclaycard orders on all products: ring Penn (Ø49481) 6666 any time of day or night. It is essential to quote your membership number to qualify for the members' discount.

HINT WINNERS

This month the £1Ø prize goes to P.Watts and the £5 prize goes to P.Wells. We are always pleased to receive more Electron hints.

MAGAZINE CASSETTE

All the programs from this month's magazine are available on cassette, and full order details appear on the back cover of ELBUG.

ELBUG MAGAZINE

GENERAL CONTENTS

HINTS, TIPS & INFO

PROGRAMS

SYNCHRONISING WORDS AND MUSIC

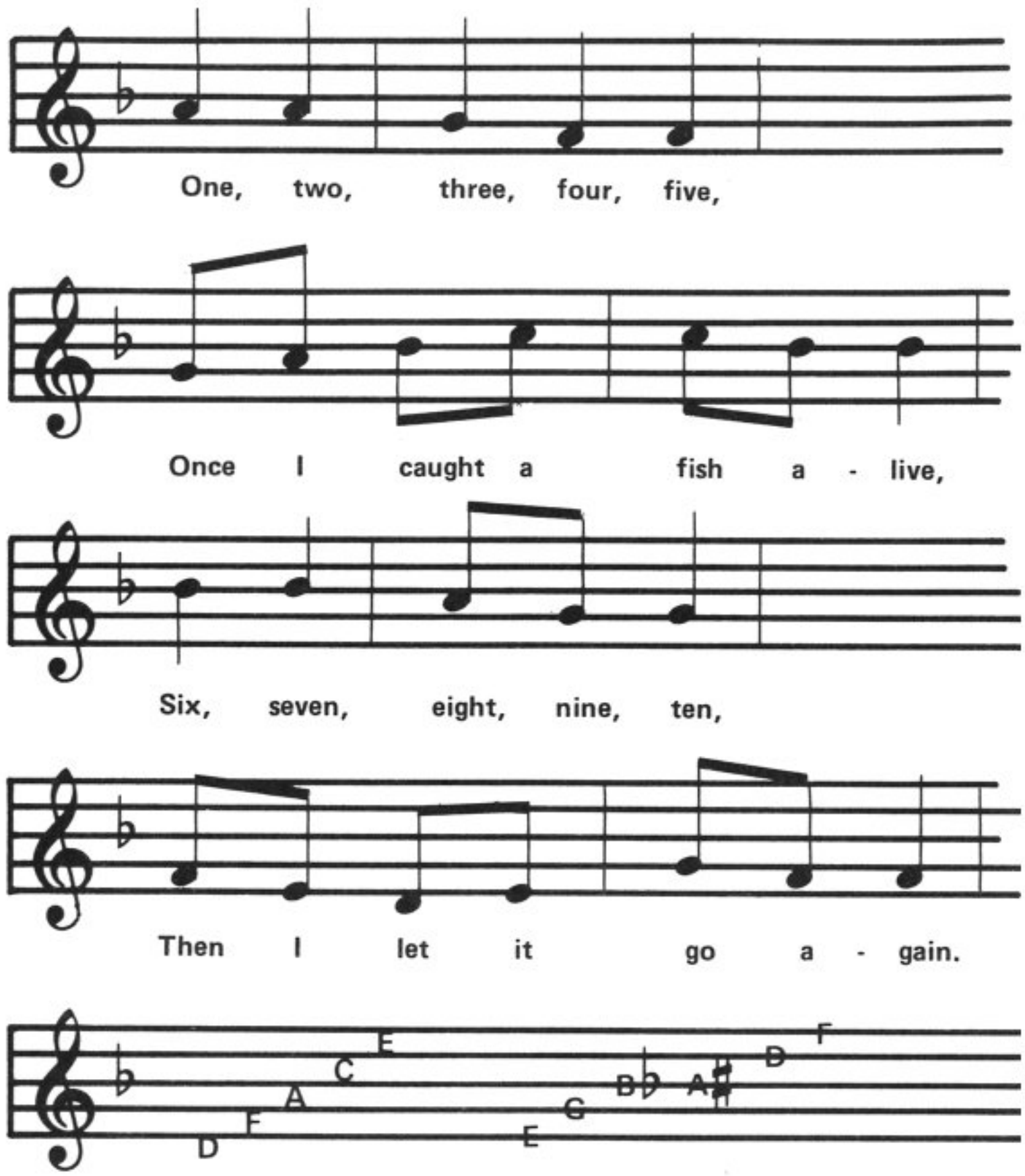
by Brian Boyde-Shaw

The Electron's ability to play music whilst it is performing other actions is very useful, but it can make it difficult to synchronize the music with the appearance of words on the screen. Brian Boyde-Shaw gives the low down on putting words to your music.

Trying to print out the words of a song while it plays through your Electron's speaker is not an easy task. Because of the way in which sound commands are handled by the Electron, there is a tendency for the words and the music to become totally separated.

In the program in this article the sound is broken down into single notes, and combined with text - words or part words - so that the text is being written on the screen at the same time as you hear the corresponding note.

The aim of the program is to play a simple nursery rhyme, one note at a time as the words are displayed on the screen.

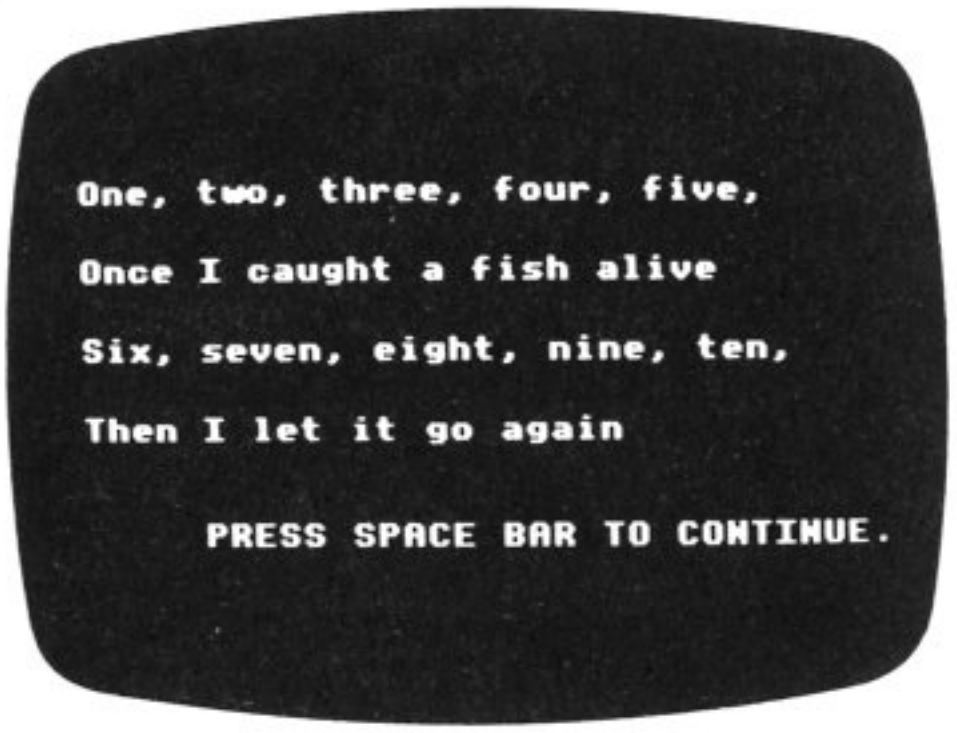


The nursery rhyme we have chosen here is 'One, two, three, four, five.', with the words and music as in the diagram. The methods used, however, can easily be adapted to any musical piece.

Let's first look at a procedure to carry out this routine. As there are twenty four notes in the rhyme, we can conveniently read in the word, or part word, the pitch of the note, and the note's duration, from DATA statements.

```
 4000 DATA one,88,8,two,88,8,three,80,4
,four,72,4,five,72,8
 4010 DATA once,80,4,I,88,4,caught,92,4
,a,100,4,fish,100,4,a,92,4,live,92,8
 4020 DATA six,92,8,seven,92,8,eight,88
,4,nine,80,4,ten,80,8
 4030 DATA then,72,4,I,68,4,let,60,4,it
,68,4,go,80,4,a,72,4,gain,72,8
```

We can then print the word in the centre of the screen, play the note, and clear the screen. Then do it all again until we reach the end of the rhyme.




```
1000 DEF PROCwords
1010 FOR R=1 TO 24
1020 READ WORD$,PITCH,DUR
1030 PRINT TAB(15,15)WORD$
1040 SOUND 1,-15,PITCH,DUR
1050 T%=TIME:REPEAT UNTIL TIME>T%+5*DUR
1060 CLS
1070 NEXT R
1080 ENDPROC
```

This procedure is designed to work in mode 4, but could be adapted for any other mode, providing that the parameters of the TAB statement (line 1030) are adjusted according to the appropriate line width.

Line 1050 will need some explanation. We need a delay loop to display the word on the screen for the time that the note sounds. If we allowed the program to continue without this delay loop, the screen would clear, then the next word would be printed, and so on, and so on, until all the words had been displayed, but the music would have hardly started. On the Electron, sounds are placed in a queue ready for processing. This takes up very little of the computer's time. Drawing and printing onto the screen, however, does take a while, so the two can get out of step. You can try this procedure without this line to see (and hear) this effect.

We must delay the printing of the 'next' word until the note has finished playing, and line 1050 does this. As the duration of the note (DUR from the data statements) is in twentieths of a second - for the SOUND statement - this delay must be converted to the units of hundredth of a second - the units of TIME. Line 1050 makes the conversion and delays the next word from appearing on the screen.

For the first note, for example, the SOUND statement requires, a duration value of 8. The delay loop, however, requires a value of 40 hundredths of a second. Line 1050 makes the conversion with the formula:

```
DELAY=DUR*5
```

You could, of course, put both timing codes separately into the data statements, but this would mean more effort typing in the music, and be wasteful of memory into the bargain.

We now have a simple procedure to show how words and music can be connected, but there are other more exciting ways of doing this. We can allow the text to build up on the screen, word by word, until the complete verse is displayed.

The same data statements can't be used because we need punctuation on the screen, now that we are to have the whole verse visible at once. When we include punctuation in data we have to wrap the data in inverted commas. Another snag is that we have to provide some means of telling the computer to start a new line for each line of the verse and that 'alive' and 'again' are two notes each but only one word. We need to introduce a coding system into our data.

```
 4050 DATA "*One,",88,8,"two,",88,8,"th
ree,",80,4,"four,",72,4,"five,",72,8
 4060 DATA "*Once",80,4,"I",88,4,"caugh
t",92,4,"a",100,4,"fish",100,4,"a",92,4
,"-live",92,8
 4070 DATA "*Six,",92,8,"seven,",92,8,"
eight,",88,4,"nine,",80,4,"ten,",80,8
 4080 DATA "*Then",72,4,"I",68,4,"let",
60,4,"it",68,4,"go",80,4,"a",72,4,"-gai
n",72,8
```

The asterisk signals that a new line is to be printed and the dash that this word is to be joined onto the previous one. This is the procedure that decodes these characters and prints out the whole verse in time to the music.

```
 2000 DEF PROClines
 2010 PRINT '''
 2020 FOR R=1 TO 24
 2030 READ WORD$,PITCH,DUR
 2040 CONTROL$=LEFT$(WORD$,1)
 2050 IF CONTROL$<>"-" AND CONTROL$<>"*
" THEN 2090
 2060 WORD$=RIGHT$(WORD$,LEN(WORD$)-1)
 2070 IF CONTROL$="*" THEN PRINT ''
 2080 IF CONTROL$="-" THEN VDU 8
 2090 PRINT WORD$;" ";
 2100 SOUND 1,-15,PITCH,DUR
 2110 T%=TIME:REPEAT UNTIL TIME>T%+5*DUR
 2120 NEXT R
 2130 PRINT  TAB(5,20)"PRESS  SPACE BAR
TO CONTINUE."
 2140 REPEAT UNTIL GET=32
 2150 ENDPROC
```

This second procedure is for mode 4 as well. Again, it could easily be converted to other modes by altering the TAB statements and the new line control characters in the data.

Line 2040 removes the control character from the lyric and checks to see if it is in fact a control character. If it isn't then the note is sounded, the word is printed onto the screen, and a delay loop used as before. This time the word is printed to follow directly on from the last one, without clearing the screen.

The lines 2070 and 2080 act according to the control character found. If an asterisk is found a new line is printed before the word. If a dash is there then the cursor is moved back one position, using VDU 8, to join the word to the last one printed.

As we are writing the complete rhyme now, clearing the screen as soon as the last word is printed isn't a good idea. A chance is given for the user to read the verse with a pause in lines 2130 and 2140. GET reads the keyboard and returns the ASCII code for the key being pressed. The space bar has an ASCII code of 32.

We now have two procedures and two sets of data, one to print out the words only, and one to print out the verse, line by line. So now let's put them into a complete program.

```
   10 MODE 4
   20 VDU23,1,0;0;0;0;
   30 REPEAT
   40 PROCcontinue
   50 UNTIL FALSE
   60 END
   70 :
 3000 DEF PROCcontinue
 3010 CLS
 3020 VDU19,0,4;0;19,1,3;0;
 3030 PRINT  TAB(5,5)"Press  W  to play
the words."
 3040 PRINT TAB(7,7)"or P to  play  the
poem."
 3050 A$=GET$
 3060 A=INSTR("WwPp",A$)
 3070 IF A=0 THEN 3050
 3080 CLS
 3090 IF A<3 RESTORE 4000:PROCwords
 3100 IF A>2 RESTORE 4050:PROClines
 3110 ENDPROC
```

First mode 4 is set up without a cursor and then the procedure, PROCcontinue is called again and again until you tire of the rhyme.

PROCcontinue simply clears the screen and redefines the foreground and background colours to prettier variants, then offers you the choice between seeing the rhyme one word at a time or building up on the screen. Your answer is compared in line 3060 with the possible legitimate answers. Whichever you pick, the right data statements are selected with the restore command and the requisite procedure called.

When you have heard 'One, two, three, four, five' to your hearts content, you may like to enter your own music. You can put any tune in as data. Each note should be entered as: word printed, pitch, and duration. Don't forget to change the total number of notes (24 in the example) in lines 1010 and 2020.

HINTS HINTS HINTS HINTS HINTS HINTS HINTS HINTS HINTS HINTS

QUICKER ENDPROCS - M.R. Bowers

When typing in ENDPROC, it is useful to know that it can be abbreviated to 'E.'.

DOWNHILL SKI RACER

by Thomas Seddon

With winter now approaching, some of you might like to try your hand at winter sports and experience the thrills (and spills) of downhill ski racing. We provide a program that is enjoyable to play, very cheap and yet free from the dangers of broken limbs.

Downhill Ski Racer is a fast action game in which you have to negotiate a series of gates on your way down the ski slopes of St. Elkwitz. There are 9 different runs in this game, each one requiring dexterity and speed of thought.

The game is very simple to play but very compulsive. You have one control - the space bar. Pressing this changes the direction of the skier as he (she) comes down the slope. You have to guide the skier through the gates of the slalom course to the final line.

If you hit a flag, marking the gates, you will not slow down as you would in the real life ski run, but such travesties are noticed and displayed at the end. More realistic is this game's treatment of hitting the trees. This is fatal!

The different runs range, from the simple slopes of the beginners slalom, with 20 gates, to the perilous precipices of the ultimate slalom, with 50 gates. At the end of each run you are given the number of gates that you hit and the number that you've passed through.

Downhill Ski Racer can be played in either mode 1 or mode 5. The choice of which run you take determines the mode. Playing in mode 1 is advisable until you become really proficient skier. The gates are further from one another in mode 1 runs, which makes steering between them easier. In mode 5 things get more difficult, especially on the 'Ultimate Slalom'.

PROGRAM NOTES

The program is reasonably well structured with the bulk of the action called from lines 110 to 400. PROCselect displays the instructions on the screen and allows you to choose your slalom run. PROCsetup does just that. It uses the course that you have chosen to select the data for that course from the series of DATA statements between lines 1570 and 1920.

PROCstart continues the setting up by preparing the screen for your race. The colours are changed and the cursor removed. Finally it waits for your signal to commence.

The actual run down the slalom is taken care of by the main section of program between lines 190 and 400. Line 220 prints out the trees at the edge of the course as the screen scrolls upwards. The screen is actually scrolled by line 380. This prints a space at the bottom right hand corner of the screen, forcing the screen to scroll upwards. The value of 'xpoke' determines at what position across the screen this is done, to accomodate the different positions of the bottom right corner in the different modes used.

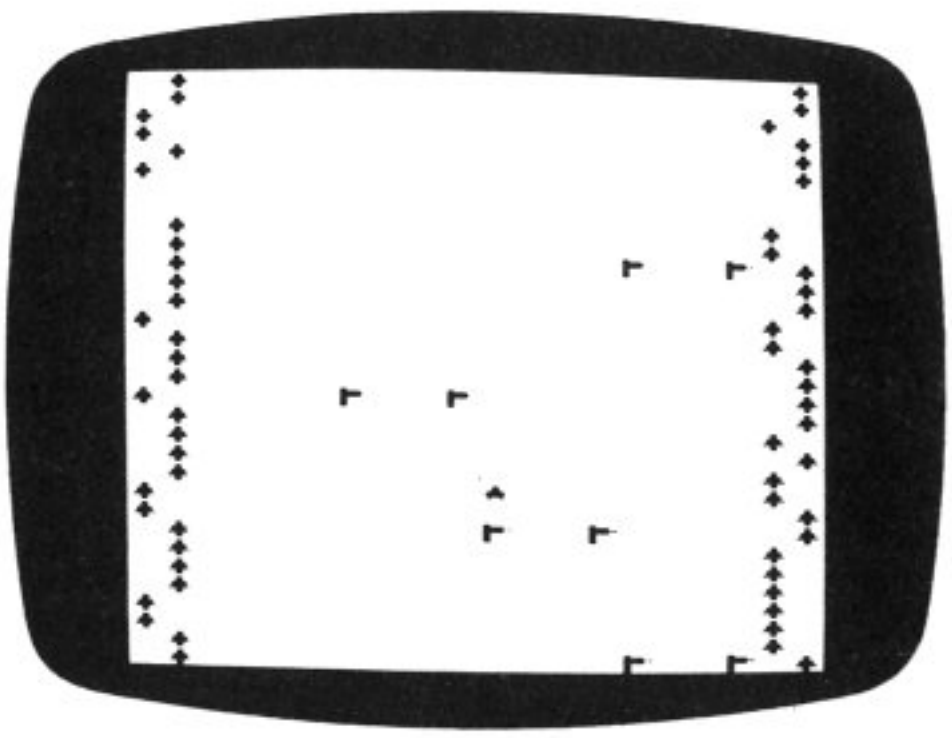
Pressing the space bar is detected in line 300 and the suitably directioned ski figure is chosen accordingly.

Detecting if you've hit anything - flag or tree - is handled by the slightly strange looking function,

FNreadcharacter, at line 1000. This uses the operating system call to read the ASCII value of a character anywhere on the screen, in this case at the position of the skier. The ASCII value is used to determine whether it is a flag or a tree that you have hit.

If it is merely a flag that you have collided with, the loop carries on and only the total of flags that you have hit is affected (line 370). If however you have hit a tree, then a procedure, PROCendofrun, is called to round the whole thing off. This procedure is also called when you have reached the end of the slalom in one piece.

PROCendofrun just asks the usual question as to whether you try again. The position of the prompt for this question, again has to be dependent on the mode being used for the run.



```
  10 REM PROGRAM SKIER
  20 REM VERSION E0.2
  30 REM AUTHOR  T.SEDDON
  40 REM ELBUG   NOVEMBER 1984
  50 REM PROGRAM SUBJECT TO COPYRIGHT
  60 :
 100 ON ERROR GOTO 2460
 110 MODE 1
 120 PROCselect
 130 :
 140 REPEAT
 150 PROCsetup
 160 MODE mode
 170 PROCstart
 180 :
 190 REPEAT
 200 co%=co%+1
 210 COLOUR 2
  220 PRINTTAB(-1+RND(tree%),29);CHR$23
4;TAB((xpoke-tree%)+RND(tree%),29);CHR$
234;
  230 COLOUR 3
  240 IF co% MOD DI%=0 AND  done%=FALSE
 READ Z%:IF Z%<>999 PRINTTAB(Z%,29);GAT
E$;:ga%=ga%+1
  250 IF Z%=999 AND done%=FALSE PRINTTA
B(tree%,29)CHR$236;STRING$(w%,CHR$235);
:COLOUR1:PRINT"FINISH";:COLOUR3:PRINTST
RING$(w%,CHR$235);CHR$233;:done%=TRUE
  260 COLOUR 1
  270 IF left% PRINTTAB(X%+1,14);" ";
  280 IF right% PRINTTAB(X%-1,14);" ";
  290 PRINTTAB(X%,15);CHR$(SKI);
  300 k%=INKEY(0)
  310 IF k%=32 AND left% right%=TRUE:le
ft%=FALSE:SKI=231:GOTO 330
  320 IF k%=32 AND right% left%=TRUE:ri
ght%=FALSE:SKI=230:GOTO 330
  330 IF left% X%=X%-1:IF X%<0 X%=0
  340 IF right% X%=X%+1:IF X%>xpoke X%=
xpoke
  350 R%=FNreadcharacter(X%,16)
  360 IF R%=138 PRINT:PRINT:SOUND 0,-15
,4,20:PROCendofrun:UNTIL TRUE:UNTIL FAL
SE
  370 IF R%=136 SOUND 1,2,100,2:fl%=fl%
+1
  380 PRINTTAB(xpoke,31);" ";
  390 IF FNgatecheck g=g+1
  400 UNTIL R%<>32 AND R%<>136
  410 :
  420 PROCdispscore
  430 PROCendofrun
  440 UNTIL FALSE
  450 END
  460 :
 1000 DEF FNreadcharacter(F,G)
 1010 VDU31,F,G
 1020 A%=135
 1030 =(USR(&FFF4) AND &FF00)DIV &100
 1040 :
 1050 DEF PROCsetupcharacters
 1060 VDU 23,230,24,24,60,126,61,36,72,
144:REM SKIER LEFT
 1070 VDU23,231,24,24,60,126,188,36,18,
9:REM SKIER RIGHT
 1080 VDU 23,232,96,124,127,124,96,96,9
6,96:REM POLE OF FLAG
 1090 VDU23,233,16,28,30,144,255,255,25
5,255:REM RIGHT END OF LINE
 1100 VDU23,236,16,28,30,17,255,255,255
,255:REM LEFT END OF LINE
 1110 VDU 23,234,0,24,24,60,60,126,24,2
4:REM FIR TREE
 1120 VDU23,235,0,0,0,0,255,0,0,0
 1130 GATE$=CHR$232+"  "+CHR$232
 1140 ENVELOPE 2,129,-15,-8,-3,10,10,10
,0,0,0,0,0,0
 1150 ENDPROC
```

```
1160 :
1170 DEF FNgatecheck
1180 C%=FNreadcharacter(X%-1,15)
1190 D%=FNreadcharacter(X%-2,15)
1200 E%=FNreadcharacter(X%+1,15)
1210 F%=FNreadcharacter(X%+2,15)
1220 IF C%=136 AND F%=136 =TRUE
1230 IF D%=136 AND E%=136 =TRUE
1240 =FALSE
1250 :
1260 DEF PROCselect
1270 PRINT
1280 PRINTTAB(12);"DOWNHILL SKIING"
1290 COLOUR 1
1300 PRINTTAB(12);"---------------"
1310 COLOUR 3
1320 PRINT:PRINT
1330 PRINT"  CONTROL YOUR SKIER BY PRE
SSING THE    SPACE BAR TO CHANGE DIRECT
ION.YOU MAY   SELECT VARYING COURSES - "
 1340 PRINT " YOU MUST GO BETWEEN THE F
LAGS AND NOT  HIT THE TREES.AT THE END
OF THE RUN THE NUMBER OF GATES YOU PASS
ED SUCCESSFULLY IS DISPLAYED ALONG WITH
 THE NUMBER OF   FLAGS KNOCKED DOWN."
 1350 PRINT:PRINT
 1360 VDU19,2,2,0,0,0
 1370 COLOUR 0:COLOUR 130
 1380 PRINT STRING$(39," ")
 1390 PRINT "1. BEGINNER'S SLALOM - 20
GATES        "
 1400 PRINT "2. INTERMEDIATE SLALOM - 2
0 GATES      "
 1410 PRINT "3. DIFFICULT SMALL SLALOM
- 20 GATES   "
 1420 PRINT "4. LONG GIANT SLALOM(HARD)
 - 30 GATES  "
 1430 PRINT "5. KING SLALOM (V.DIFFICUL
T) - 50 GATES"
 1440 PRINT "6. FAST SLALOM(V.DIFFICULT
) - 20 GATES "
 1450 PRINT "7. FAST GIANT SLALOM(HARD)
 - 20 GATES  "
 1460 PRINT "8. FAST TREE HUGGER(HARD)
- 20 GATES   "
 1470 PRINT "9. THE ULTIMATE SLALOM - 5
0 GATES      "
 1480 PRINT STRING$(39," ")
 1490 COLOUR 3:COLOUR 128
 1500 PRINT:PRINT
 1510 PRINTTAB(11,30);"WHICH DO YOU WAN
T"
 1520 REPEAT:W$=GET$:UNTIL VAL(W$)>0
 1530 dp=1630
 1540 FOR A=1 TO VAL(W$):dp=dp+30:NEXT
 1550 ENDPROC
 1560 :
 1570 REM COURSE DATA
 1580 REM
 1590 REM 1st NUMBER IS VERTICAL
 1600 REM DISTANCE BETWEEN GATES AND
 1610 REM FOLLOWING ARE DISTANCE OF
 1620 REM NEXT GATE FROM THE LEFT EDGE
 1630 REM THE STATEMENT ENDS WITH 999
 1640 :
 1650 REM BEGINNERS
 1660 DATA 8,1,39,15,19,10,14,10,9,8,10
,15,17,20,22,23,27,29,31,29,25,22,20,999
 1670 :
 1680 REM INTERMEDIATE
 1690 DATA 8,1,39,19,20,18,20,22,20,18,
16,14,16,18,20,22,24,26,28,26,24,22,20,
999
 1700 :
 1710 REM SMALL DIFFICULT
 1720 DATA 4,1,39,18,22,18,22,18,22,18,
22,18,22,18,22,18,22,18,22,18,22,18,22,
999
 1730 :
 1740 REM GIANT LONG HARD
 1750 DATA 8,1,39,20,14,8,8,8,14,20,20,
20,14,8,8,8,14,20,20,20,14,8,8,8,14,20,
20,20,14,8,8,8,14,999
 1760 :
 1770 REM KING SLALOM
 1780 DATA 7,1,39,20,15,10,5,8,5,9,5,10
,5,11,5,12,19,26,32,29,32,28,32,27,32,2
6,32,25,28,21,27,20,20,20,13,6,13,7,13,
20,27,25,29,27,27,20,13,20,27,20,13,17,
20,999
 1790 :
 1800 REM FAST SLALOM
 1810 DATA 8,5,19,10,4,10,4,10,4,10,4,1
0,4,10,4,10,12,10,12,10,8,12,10,999
 1820 :
 1830 REM FAST GIANT SLALOM
 1840 DATA 6,5,19,10,4,10,4,10,4,10,4,1
0,4,10,14,10,14,13,12,11,10,4,10,999
 1850 :
 1860 REM FAST TREE HUGGER
 1870 DATA 5,5,19,10,5,8,3,7,3,6,3,5,3,
4,3,3,3,8,13,8,3,8,13,999
 1880 :
 1890 REM FAST KING SLALOM
 1900 DATA 7,5,19,10,14,6,10,14,10,6,10
,14,10,6,10,3,10,3,9,3,8,3,5,3,3,10,14,
7,14,7,14,7,14,7,14,7,14,7,14,7,14,7,14
,7,14,15,15,8,3,8,10,14,10,999
 1910 :
 1920 REM END OF COURSE DATA
 1930 :
 1940 DEF PROCsetup
 1950 RESTORE dp
 1960 READ DI%
 1970 READ mode
 1980 IF mode=1 tree%=4 ELSE IF mode=5
tree%=2
 1990 READ xpoke
 2000 IF mode=1 width=15 ELSE width=5
 2010 Z%=0
```

```
 2020 co%=0:ga%=0
 2030 g=0
 2040 fl%=0
 2050 done%=FALSE
 2060 SKI=230:IF mode=1 X%=19 ELSE IF m
ode=5 X%=9
 2070 PROCsetupcharacters
 2080 left%=TRUE
 2090 right%=FALSE
 2100 ENDPROC
 2110 :
 2120 DEF PROCstart
 2130 VDU23,1,0;0;0;0;
 2140 IF mode=1 w%=12 ELSE IF mode=5 w%
=4
 2150 VDU19,0,7,0,0,0,19,3,1,0,0,0,19,2
,2,0,0,0,19,1,4,0,0,0
 2160 PRINTTAB(tree%,13);CHR$236;STRING
$(w%,CHR$235);:COLOUR 1:PRINT"START";:C
OLOUR3:PRINTSTRING$(w%,CHR$235);CHR$233;
 2170 COLOUR 1:PRINTTAB(((xpoke+1)/2)-1
0,5);"PRESS SPACE TO START";:REPEAT UNT
IL GET=32:PRINTTAB(((xpoke+1)/2)-10,5);
SPC20;
 2180 ENDPROC
 2190 :
 2200 DEF PROCdispscore
 2210 IF mode=1 ta=16 ELSE IF mode=5 ta
=6
 2220 IF g=ga% PROCclearrun:ENDPROC
 2230 PRINT:PRINT:PRINT:PRINT:PRINTTAB(
ta);g;:IF g=1 PRINT" GATE" ELSE PRINT"
GATES";TAB(ta);fl%;:IF fl%=1 PRINT" fla
g down" ELSE PRINT " flags down"
 2240 IF g=ga% PROCclearrun
 2250 ENDPROC
 2260 :
 2270 DEF PROCendofrun
 2280 PRINT:PRINT
 2290 IF mode=1 tab=8 ELSE tab=0
 2300 PRINTTAB(tab);"SAME RUN AGAIN (Y/
N)"
 2310 REPEAT:W=GET:UNTIL W=89 OR W=78:I
F W=89 GOTO 2330
 2320 PRINT:PRINT:PRINTTAB(tab)"A DIFFE
RENT RUN THEN (Y/N)":REPEAT:W=GET:UNTIL
 W=89 OR W=78:IF W=89 RUN:ELSE VDU22,6:
END
 2330 ENDPROC
 2340 :
 2350 DEF PROCclearrun
 2360 PRINT:PRINT:PRINT:PRINT:PRINTTAB(
ta-4);"ALL GATES PASSED";
 2370 PRINT:PRINT
 2380 RESTORE 2430
 2390 REPEAT
 2400 READ P,D
 2410 SOUND 1,-15,P,D
 2420 UNTIL P=151
 2430 DATA 100,2,150,2,100,2,150,2,200,
2,250,2,200,2,150,2,200,2,180,2,160,2,1
40,2,120,2,100,2,80,2,120,2,160,2,200,2
,240,2,230,2,220,2,200,2,175,2,170,2,16
5,2,160,2,155,2,150,2,100,2,50,2,1,2,50
,2,100,2,50,2,100,2,50,2,100,2,151,2
 2440 ENDPROC
 2450 :
 2460 ON ERROR OFF:MODE 6
 2470 IF ERR<>17 REPORT:PRINT" at line
";ERL
 2480 END
```

HINTS HINTS HINTS HINTS HINTS HINTS HINTS HINTS HINTS

FASTER BASIC - P.J. Vincent

When performing certain operations in Basic, some programming techniques are more efficient than others (timewise). Generally speaking, REPEAT-UNTIL is faster than IF-THEN-GOTO, but FOR-NEXT is faster than REPEAT-UNTIL. A GOSUB is slightly faster than a procedure, contrary to what the User Guide might suggest.

BASICALLY MISTAKEN - Roger Garratt

Basic gets confused in assembler with variable names that begin with a capital 'A'; it thinks you want Accumulator addressing, and assembles accordingly. There are two basic solutions: use lower case variable names, or enclose the variable in brackets.

AMUSING *FX CALL - W.J. Hicks

Try *FX243,n where n is a number between 0 and 255. Some, for example 69, 79, 200 and 255, can have some very amusing results. Press the Break key or Return a few times to produce more effects, or experiment with other values! Control-Break will reset the machine afterwards.

A UNIVERSAL JOYSTICK ROUTINE FOR THE PLUS 1

by Jason Kristiansen

Many Electron owners will have bought, or be thinking of buying Acorn's Plus-1 expansion board. The majority of Electron games programs, however, do not have an joystick option available. This short utility will adapt most Basic and machine code programs to run with a joystick.

The Analogue Joystick Adaptor Routine (AJAR for short) will enable you to use an analogue joystick and Plus-1 interface instead of a set of keyboard keys in the vast majority of Basic programs and most commercial machine code games. When you have typed in the program, save it onto cassette before you run it. A small mistake in your typing could well not only stop the program from working but lose the entire program as well.

MENU

When you do run the program you are presented with a menu of options. The most important of these are the two (I and D) that actually set up the machine code to use the joysticks. There are two methods of reading the Electron's keyboard - using INKEY(-ive) and INKEY(+ive). Machine code games use machine code equivalents of these as well. AJAR can cope with programs using either or both methods. There are however two separate routines, one for each. Choosing the indirect option (I) will set up the code for using joysticks when the indirect key-reading method (INKEY(+ive)) is used. The option, D, will set up for direct, or INKEY(-ive), key reading.

If your game program is in Basic then you should use the routine to suit the method used in the program. If, however, you want to use joysticks with a machine code game, a bit of experimentation is needed. The majority of commercial software uses the direct method (D) but some (of course) do not.

Choosing either of these options allows you to choose which key each joystick action will emulate. This done, the code is assembled and it's back to the menu.

Once either or both sets of code have been set up, you can exit the program (another menu option) and load your own Basic or machine code program. This will now work with the joysticks.

POSITION OF THE ROUTINES

Another option on the menu is one to change the position of the two machine code routines in the Electron's memory. There are default values provided in the program as it stands. These may not operate correctly with all commercial games. Option A allows you to examine and alter the position of either of the routines. Of course they must not overwrite each other if you are using both.

Routine one is &68 bytes long and normally sits at &11Ø. Routine two is &99 bytes long and normally sits at &966. &11Ø and &966 were chosen because the routines are unlikely to be overwritten by the game at these places.

Some locations where the routine(s) could start are as follows:

Locations	Description	Uses
&110-&186	Top of stack	Commercial games
&900-&AE0	Buffer for OPEN (IN/OUT/UP)	Basic without OPEN
&B00-&BFF	Buffer for function keys	Basic without f keys
&D02-&DFF	Rom cartridges Disc expansion	Basic programs

JOYSTICK SENSITIVITY.

The routines check the voltage across the A/D converter using the machine code equivalent of ADVAL. If this value rises above a certain point (if the joystick is moved far up or left) the routines will return a 'key-pressed' signal.

The opposite is true for down/right. The further apart the values are, the less sensitive the joystick is. The option is given to adjust the default values of these 'Joyhigh' and 'Joylow'.

SAVING THE ROUTINES

The final option is to save just the machine code routines generated by this program. Once you have assembled the code (using option D or I) choosing option S will allow you to save either or both of the routines to cassette under a filename of your choice. These routines can then be loaded into the machine and run ready for use with a game by typing *RUN "filename". The advantage of this is that the routines are very short and can be loaded into your Electron much quicker than the whole of AJAR.

TECHNICAL DETAILS

The procedure, PROCgetkeys, inputs the user's choice of keys and translates them into INKEY(-ve) and ASCII values. The centre of the program contains the two assembler routines, 'assemdiread' and 'assemindiread', as procedures.

PROCassemdiread works by intercepting a keyboard vector and checking joystick values. It also enables two ADC channels which might otherwise be disabled by the host program.

PROCassemindiread intercepts the interrupt vector IRQV1 at &204 to check the joystick and insert corresponding values into the keyboard buffer.

```
  10 REM PROGRAM AJAR
  20 REM VERSION E0.2
  30 REM AUTHOR  J.Kristiansen
  40 REM ELBUG   NOVEMBER 1984
  50 REM PROGRAM SUBJECT TO COPYRIGHT
  60 :
 100 ON ERROR GOTO 2810
 110 :
 120 MODE6
 130 PROCinit
 140 REPEAT:PROCmenu:UNTIL OPT$="E"
 150 PROCexitprog
 160 END
 170 :
1000 DEFPROCgetkeys
1010 KEY=&FF
1020 *FX4,1
1030 CLS:PRINT
1040 RESTORE
1050 FOR getkey=1 TO 5
1060 READ direction$
1070 PRINT"PRESS ";direction$;" KEY"
1080 *FX21,0
1090 IF KEY=64 OR KEY=0 KEY=&FF:TIME=0
:REPEAT UNTIL TIME=30
1100 A%=122:REPEAT
1110 !getxblock=USR(OSBYTE)
1120 keys(2,getkey)=INKEY(0)
1130 IF INKEY(-1) KEY=0
1140 IF INKEY(-65) KEY=64
1150 UNTIL (keys(2,getkey)<>-1 AND get
xblock?1<>255) OR KEY=0 OR KEY=64
1160 IF KEY<>0 AND KEY<>64 KEY=getxblo
ck?1
1170 keys(1,getkey)=KEY+&80
1180 NEXT getkey
1190 *FX 21,0
1200 *FX 4,0
1210 ENDPROC
1220 DATA LEFT,RIGHT,UP,DOWN,FIRE
1230 :
1240 DEFPROCassemdiread
1250 FOR PASS=0 TO 3 STEP3:P%=assem1%
1260 [OPT PASS
1270 .newkeyentry:BVS finish
1280 BCC finish
1290 .checkjoystick:PHA
1300 CMP# keys(1,1)
1310 BEQ left
1320 CMP#keys(1,2)
```

```
1330 BEQ right
1340 CMP#keys(1,3)
1350 BEQ up
1360 CMP#keys(1,4)
1370 BEQ down
1380 CMP#keys(1,5)
1390 BEQ fire
1400 .notvalidkey:SEC:CLV:PLA
1410 .finish:JMP OSKEY
1420 .left
1430 LDX#1:JSR osjoydetect
1440 CPY#joyhigh:BCS joypressed
1450 BCC nojoy
1460 .right
1470 LDX#1:JSR osjoydetect
1480 CPY#joylow:BCC joypressed
1490 BCS nojoy
1500 .up
1510 LDX#2:JSR osjoydetect
 1520 CPY#joyhigh:BCS joypressed
 1530 BCC nojoy
 1540 .down
 1550 LDX#2:JSR osjoydetect
 1560 CPY#joylow:BCC joypressed
 1570 BCS nojoy
 1580 .fire:LDA#16:LDX#2:JSR OSBYTE:\EN
ABLE 2 ADC CHANNELS
 1590 LDX#0:JSR osjoydetect
 1600 TXA:AND#3
 1610 BNE joypressed
 1620 .nojoy CLC:PLA:BCC finish
 1630 .osjoydetect:LDA#128:JMP OSBYTE
 1640 .joypressed:PLA:LDA#128:RTS
 1650 .exec1 SEI:LDA#newkeyentry MOD 25
6:STA &228:LDA#newkeyentry DIV 256:STA
&229:CLI:RTS
 1660 .fin1
 1670 ]
 1680 NEXT PASS
 1690 DIM alterdirec 50
 1700 P%=alterdirec
 1710 [SEI:LDA# newkeyentry MOD256:STA&
228:LDA# newkeyentry DIV256:STA&229:CLI
:RTS
 1720 ]
 1730 ENDPROC
 1740 :
 1750 DEFPROCaltervectors(VECTOR)
 1760 IF VECTOR=1 CALL alterdirec:alter
1=TRUE
 1770 IF VECTOR=2 ?OSFVEC=osfile2 MOD25
6:OSFVEC?1=osfile2 DIV256:CALL alterirq
:alter2=TRUE
 1780 IF alter1 AND alter2 PROCexitprog
 1790 ENDPROC
 1800 :
 1810 DEFPROCassemindiread
 1820 FOR PASS=0 TO 3 STEP3
 1830 P%=assem2%:[OPT PASS
 1840 .ENTRY
 1850 PHP:PHA
 1860 DEC fullflag:LDA fullflag:CMP#&A1
:BNE finish2
 1870 TXA:PHA:TYA:PHA:CLV
 1880 LDA#128:LDX#1:JSR OSBYTE
 1890 TYA:CMP# joyhigh:BCS left
 1900 .read2 CMP#joylow:BCC right
 1910 .read3:LDA#128:LDX#2:JSR OSBYTE
 1920 TYA:CMP#joyhigh:BCS up
 1930 .read4 CMP#joylow:BCC down
 1940 .read5 LDX#0:LDA#128:JSR OSBYTE
 1950 TXA:AND#3:BNE fire:LDA#&B2:STA fu
llflag
 1960 .finish:PLA:TAY:PLA:TAX:.finish2:
PLA:PLP
 1970 JMP IRQREQ
 1980 .left PHA:LDA#(keys(2,1))
 1990 JSR keybuffer
 2000 PLA
 2010 BVC read2
 2020 .right LDA#keys(2,2)
 2030 JSR keybuffer
 2040 BVC read3
 2050 .up:PHA:LDA#keys(2,3)
 2060 JSR keybuffer
 2070 PLA:BVC read4
 2080 .down:LDA#keys(2,4)
 2090 JSR keybuffer
 2100 BVC read5
 2110 .fire LDA#keys(2,5)
 2120 JSR keybuffer
 2130 BVC finish
 2140 .keybuffer:TAY:LDX#0:LDA#153:JMP
OSBYTE
 2150 .fullflag:EQUB&FF
 2160 .osfile2:PHA:SEI:LDA# IRQREQ MOD2
56:STA IRQV1:LDA# IRQREQ DIV256:STA IRQ
V1+1:PLA:CLI:JSR OSFILE:.alterirq SEI:L
DA#assem2% MOD256:STA IRQV1:LDA#assem2%
 DIV256:STA IRQV1+1:CLI:RTS
 2170 .exec2 LDA#osfile2 MOD 256:STA OS
FVEC:LDA #osfile2 DIV 256:STA OSFVEC+1:
JMP alterirq
 2180 .fin2
 2190 ]
 2200 NEXT PASS
 2210 ENDPROC
 2220 :
 2230 DEFPROCinit
 2240 DIM getxblock 4,keys(2,5)
 2250 OSBYTE=!&20A AND&FFFF
 2260 joyhigh=&C9:joylow=&43:REM LIMITS
 FOR JOYSTICK DETECTION
 2270 assem1%=&110:REM START ASSEMBLY L
OCATION FOR DIRECT READ ROUTINE
 2280 assem2%=&966:REM LOCATION FOR IND
IRECT READ ROUTINE
 2290 OSKEY=!&228 AND&FFFF
 2300 IRQV1=&204:IRQREQ=!&204 AND&FFFF
 2310 OSFVEC=&212:OSFILE=!&212 AND&FFFF
```

```
 2320 alter1=FALSE:alter2=FALSE
 2330 ENDPROC
 2340 :
 2350 DEFPROCmenu
 2360 CLS:PRINT'"JOYSTICK CONFIGURE UTI
LITY"'STRING$(32,"_")''
 2370 PRINT'"A-ALTER OR EXAMINE ASSEMBL
Y DETAILS"''"D-SET up CODE FOR READING
KEYS DIRECTLY"''"I-SET up CODE FOR READ
ING INDIRECTLY"''"S-SAVE ASSEMBLED CODE
"''"E-EXIT THIS PROGRAM"''
 2380 OPT$=GET$
 2390 IF OPT$="A" PROCexamine
 2400 IF OPT$="D" PROCdiread
 2410 IF OPT$="I" PROCindiread
 2420 IF OPT$="S" PROCsave
 2430 ENDPROC
 2440 :
 2450 DEFPROCexamine
 2460 CLS:PRINT'"Limits of joystick det
ection"
 2470 joyhigh=FNquest("LEFT/UP",joyhigh)
 2480 joylow=FNquest("RIGHT/DOWN",joylo
w)
 2490 PRINT''"Start assembly locations
for routines:"'
 2500 assem1%=FNquest("Direct read joys
tick routine",assem1%)
 2510 assem2%=FNquest("Indirect read jo
ystick routine",assem2%)
 2520 ENDPROC
 2530 :
 2540 DEFFNquest(quest$,var):PRINT ques
t$;":&";:xpos=POS:ypos=VPOS:PRINT TAB(x
pos,ypos);~var;:INPUT TAB(xpos,ypos)res
p$:IF resp$="" THEN =var ELSE =EVAL("&"
+resp$)
 2550 :
 2560 DEFPROCdiread
 2570 IF assem1%=assem2% OR alter1 PRIN
T'"INVALID ASSEMBLY DATA/CODE ASSEMBLED
":WAIT=INKEY(90):ENDPROC
 2580 PROCgetkeys
 2590 PROCassemdiread
 2600 PROCaltervectors(1)
 2610 ENDPROC
 2620 :
 2630 DEFPROCindiread
 2640 IF assem1%=assem2% OR alter2 PRIN
T'"INVALID ASSEMBLY DATA/CODE ASSEMBLED
":WAIT=INKEY(70):ENDPROC
 2650 PROCgetkeys
 2660 PROCassemindiread
 2670 PROCaltervectors(2)
 2680 ENDPROC
 2690 :
 2700 DEFPROCexitprog
 2710 PRINT''"Now load your program as
normal."''
 2720 END
 2730 DEFPROCsave
 2740 IF NOT alter1 AND NOT alter2 THEN
 ENDPROC
 2750 CLS:PRINT ''' "D-SAVE DIRECT CODE
"''"I-SAVE INDIRECT CODE"
 2760 ID$=GET$
 2770 IF ID$="D" AND alter1 THEN INPUT
''"FILENAME: "NM$:OSCLI "SAVE "+LEFT$(N
M$,7)+" "+STR$~(assem1%)+" "+STR$~(fin1
)+" "+STR$~(exec1)
 2780 IF ID$="I" AND alter2 THEN INPUT
''"FILENAME: "NM$:OSCLI "SAVE "+LEFT$(N
M$,7)+" "+STR$~(assem2%)+" "+STR$~(fin2
)+" "+STR$~(exec2)
 2790 ENDPROC
 2800 :
 2810 ON ERROR OFF:MODE 6
 2820 IF ERR=17 END
 2830 REPORT:PRINT" at line ";ERL
 2840 END
```

HINTS HINTS HINTS HINTS HINTS HINTS HINTS HINTS HINTS HINTS

BEEP ON ERROR - D. Constable

This short routine causes the computer to emit a short beep when an error is detected - a feature found on some other computers e.g. Apple and Acorn Atom. The program works by re-directing the BRK vector at &202 and &203 to a piece of code which prints Ctrl-G to the screen, then jumps to the normal error handling routine. To use the program, run it, and then save it with:

*SAVE ERRBEEP 0D01 0D22

Thereafter you need only type *RUN ERRBEEP to put it into effect.

```
10     FOR J%=0 TO 2 STEP 2:P%=&D01
20     [OPT J%
30     LDA &202:STA &70:LDA &203: STA &71
40     LDA #start MOD 256: STA &202
50     LDA #start DIV 256: STA &203
60     RTS
70     .start
80     PHP: PHA: LDA#7: JSR&FFEE
90     PLA: PLP: JMP (&70)]
100    NEXT:END
```

WHIRLPOOL

by D. D. Harriman

The Electron is blessed with an excellent potential for graphics displays which is shown off well by this program. Not only does it draw a impressive picture on your screen but animates it as well.

This program is not long but it is very effective. Type the program into your Electron especially carefully. It involves some quite complicated maths which will not work at all if you make just a few little mistakes. When you have typed in the program. Save it onto a cassette before you run it, just in case.

When you do run the program, a vast whirlpool is slowly drawn out to cover the whole screen, swirling and spinning all the while. Whirlpool appears three dimmensional and it is in full colour, but the colours are chosen to give a good effect with a black and white display as well as a colour one. Just when you thought it was safe to go back to the plug hole...

PROGRAM NOTES

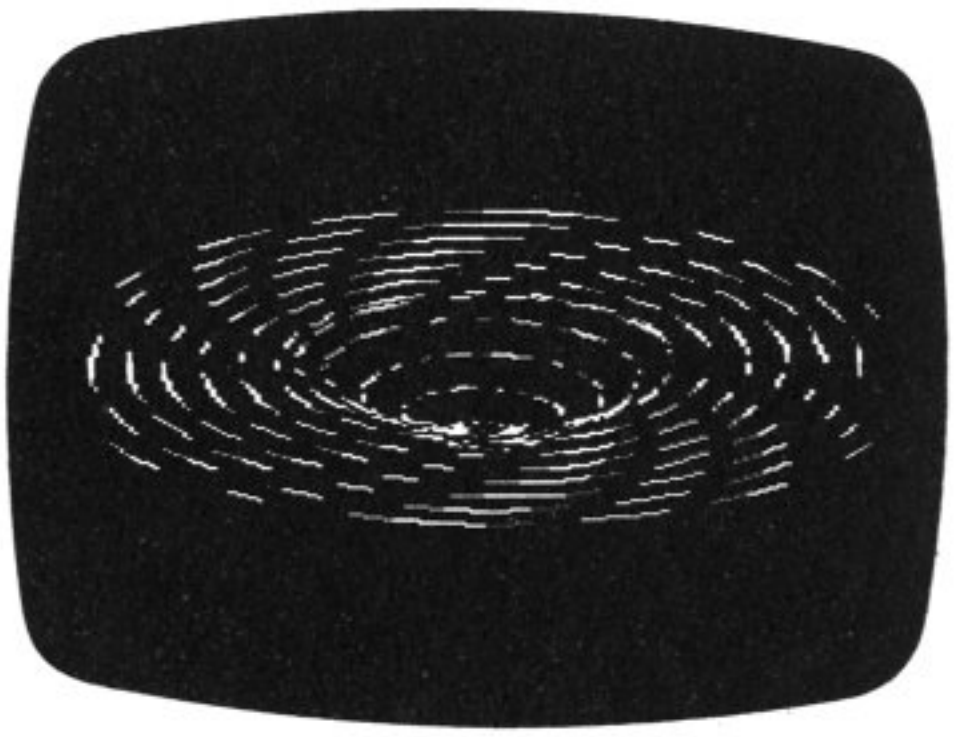
This program is in three distinct parts. The first part to be executed is a procedure, PROCsetup, that makes everything ready for the rest of the program. The most important action of this program section is to work out several cosine and sine values. These are put into the arrays, cos and sin. The Electron is fairly slow at calculating trigonometrical values. As the main part of the program uses these values several times each, it speeds up the drawing considerably to calculate them in advance. This calculating takes quite a while so a 'please wait' message is put up on the screen to reassure you.

PROCsetup also does such housework as switching off the cursor and clearing the screen.

The main part of the program uses two procedures, PROCspiralone and PROCspiraltwo, to draw the two parts of the whirlpool using the trig values obtained earlier. As these spirals are drawn out, the colour used for drawing is constantly changed, cycling through the sixteen colours available in the mode two used. The colours are being constantly redefined, using the VDU19 command, by the procedure, PROCcolour, to give the illusion of a moving picture.

This is another example of the technique of animating graphics using the VDU19 command as described in the last part (October ELBUG) of our Electron Graphics series.

```
  10 REM PROGRAM WHIRLPOOL
  20 REM AUTHOR  D.D.Harriman
  30 REM ELBUG   NOV 1984
  40 REM PROGRAM SUBJECT TO COPYRIGHT
  50 :
  60 ON ERROR GOTO 1470
  70 :
 100 MODE 2
 110 PROCsetup
 120 PROCspiralone
 130 PROCspiraltwo
 140 REPEAT
 150 PROCcolour
 160 UNTIL FALSE
 170 END
 180 :
1000 DEF PROCsetup
1010 PRINT TAB(5,5)"SETTING UP"TAB(3,8
)"Please wait..."
1020 DIM C%(8)
1030 DIM sin(62),cos(62)
1040 FOR F%=1 TO 8:READ C%(F%):NEXT F%
1050 B%=0
1060 FOR F=0 TO PI*2 STEP PI/30
1070 sin(B%)=SIN F:cos(B%)=COS F
1080 B%=B%+1
1090 NEXT F
1100 C%=0
1110 VDU23,1,0;0;0;0;
1120 CLS
1130 ENDPROC
1140 :
1150 DEF PROCspiraltwo
1160 B%=300
1170 FOR I%=1 TO 20
1180 FOR A%=0 TO 60
```



```
1190 PROCcolour
1200 B%=B%+1:GCOL0,B% MOD 15+1
1210 DRAW 640+cos(A%)*B%,512+sin(A%)*B
% DIV 3
1220 NEXT A%
1230 NEXT I%
1240 ENDPROC
1250 :
1260 DEF PROCspiralone
1270 B%=51
1280 MOVE 680,400
1290 FOR I%=1 TO 4
1300 FOR A%=0 TO 60
1310 PROCcolour
1320 B%=B%+1:GCOL0,B% MOD 14+1
1330 X%=648+cos(A%)*B%
1340 Y%=370+B% DIV 2+sin(A%)*B% DIV 3
1350 DRAW X%,Y%
1360 NEXT A%
1370 NEXT I%
1380 ENDPROC
1390 :
1400 DEF PROCcolour
1410 C%=C%+1
1420 FOR G%=1 TO 8
1430 VDU 19,(C%+G%) MOD 15+1,C%(G%);0;
1440 NEXT G%
1450 ENDPROC
1460 :
1470 ON ERROR OFF
1480 MODE 6
1490 IF ERR<>17 REPORT:PRINT " at line
";ERL
1500 END
1510 :
1520 DATA 0,4,1,5,2,6,3,7
```

NEWS NEWS NEWS

MODE 7 FOR THE ELECTRON

You've probably heard of the promise of an add-on, from Sir Computers, to give your Electron the much vaunted mode 7 of the BBC micro. Sir displayed a prototype of the mode 7 adaptor at the recent Acorn User Show. Unfortunately it seems that Sir bit off more than it can, at least for the moment, chew. Problems with the software mean the adaptor won't be on sale for another few weeks at the earliest. You can grumble at Sir on 0222-621813.

TALK AMONGST YOURSELVES

If you are dying to enter the world of 'comms' and 'hacking', you will be pleased to hear that Protek Computing (0506-415353) are working on an interface for the Electron that will allow you to connect most standard modems to your computer. The interface will be out at the end of the year and retail for £24.95. This will include software to run Protek's own modem (£59.95) to both access Prestel and operate in a 'user to user' mode.

GO FORTH

If you fancy trying your hand at Forth and you have one of the Electron Rom expansion boards then Skywave software (0202-302385) has the Rom for you. Multi-Forth 83 is a complete implementation of the Forth 83 standard for the Electron. Not only does this allow you to enjoy the delights of the Forth language but it also gives your Electron multitasking - the ability to run several (Forth) programs at once. Multi Forth 83 costs £48. You'll find Skywave at 73 Curson Road, Bournemouth, BH1 4PW.

BBC GOES ELK

BBCsoft is now releasing software for the Electron and converting many of the existing BBC micro range. The much acclaimed White Knight chess package is amongst the titles. Others include Wordmover (a text editor), Locomotion (a graphics adventure type game), and 'Maths-with a story!' (an educational piece). All these are £9.95 except Locomotion which is £6.95. Further conversions are on the way.

POWERSOFT JOYSTICK INTERFACE

Reviewed by Geoff Bains

There are now several joystick interfaces available for the Electron. Powersoft is fairly late to enter the race. Geoff Bains sees what extra it has to offer.

Product : Joystick interface
Price : £24.95
Supplier : Power software,
12 Hagley Road,
Stourbridge,
DY8 1PS.
(0384) 370811

Most of the joystick interfaces that you can buy for your Electron are either very expensive, very inconvenient to use, or both. Powersoft has overcome these problems with its new joystick interface. The Powersoft interface enables you to use joysticks with any Basic or machine code game instead of the keyboard.

The Powersoft model comes in a small plastic case, coloured the inevitable cream. On one side of this there is a socket to connect the interface to the Electron's expansion port and on the other a single standard Atari type joystick socket.

This interface, like all the joystick interfaces for the Electron, except Acorn's own Plus-1, is for switch type joysticks. These are not as good as the analogue type for some serious applications but are ideal for playing arcade games. It is a shame that two joystick sockets could not have been included. As it is the Powersoft interface will be useless for two player games (though these are admittedly very thin on the ground). It is also regrettable that there is no connector to allow further expansions to be plugged into this interface.

Despite these quibbles, the Powersoft interface wins hands down for convenience. When you switch on your Electron, as well as the normal 'Acorn Electron 32K' message there now appears 'Powersoft Joystick Interface'. At this moment the joystick will do nothing. Type *Joy and the interface's built in software will present you with a series of menu questions to allow you to configure your joystick to take the place of any set of keys. There's no waiting for cassettes to load the interface's software.

There are some nice touches to the software, too. If you try to define a single joystick action to take the place of two keys (say, the fire button operating both the laser and the bombs in 'Moonraider'), the software checks that you do in fact mean that. When you've finished the complete configuration, a further check is made that all is correct before going on.

Once you have defined which keys are going to be duplicated by the joystick, your Electron returns to normal. Normal that is, except that any software reading the keyboard will now unwittingly read the joystick port as well. Now you can load your favorite game, or write yourself a new one, and away you go, joystick in hand. One advantage of the Powersoft interface over others around is that you can always switch back to using keys in the middle of a game because the two are running in parallel.

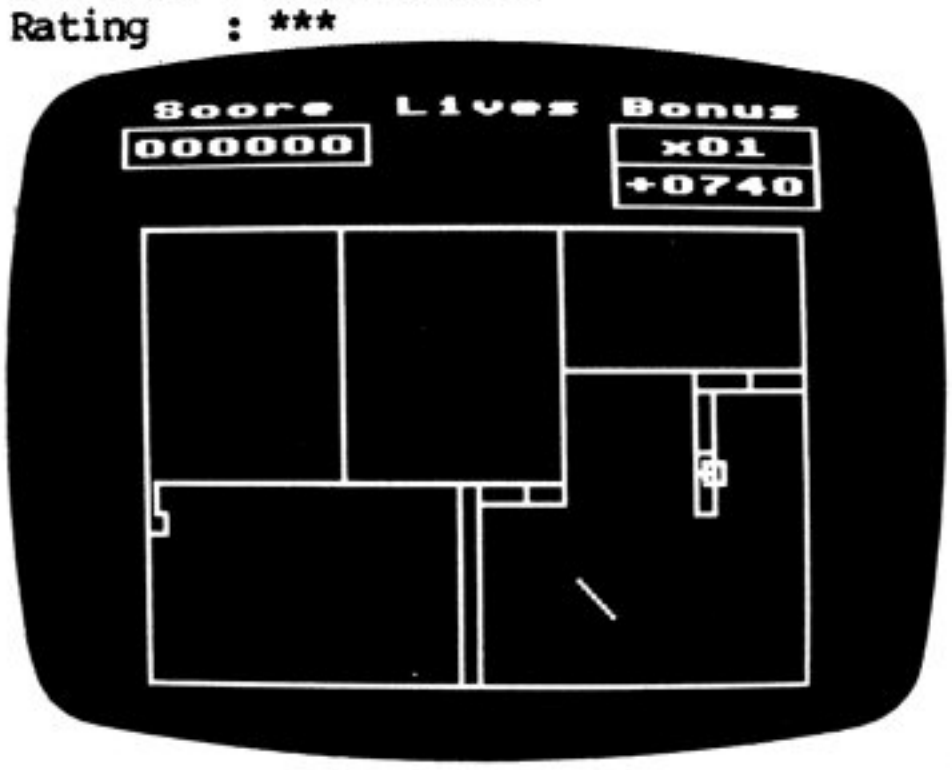
There are in fact two distinct methods of reading the Electron's keyboard that commercial machine code software uses. The Powersoft interface

can cope with either of these but you have to say which it is that your game is using when defining the key set at switch-on. As a game that uses one method won't be able to use the interface configured for the other method, it can be a time consuming and tedious affair to find out how to configure the interface for any particular game. Fortunately most games use method one (as Powersoft calls it) only. In any case Powersoft is willing to sort out any difficult cases for you if you run into problems.

The Powersoft interface is certainly good value for money. It will give you joysticks with your Electron in a more convenient way and for less money than anything else anything else currently on the market.

SOFTWARE FOR GAMES PLAYERS

Name : Frenzy
Supplier : Micro Power
Price : £6.95
Reviewer : Geoff Bains
Rating : ***



Frenzy is a deceptively simple game that will keep you happily frustrated for hours. Of course the game is dressed up in the now-standard sci-fi scenario: You control a robot craft in a scientific research centre in an attempt to destroy the stray lepton particles (a sort of high speed caterpillar) that are darting around the building. Your craft leaves an 'ion' trail. When you enclose an area of the building with the trail it is filled in and, if the lepton is inside that area, it is destroyed. Unfortunately the filling operation takes a tediously long time on the sedate Electron. If a lepton touches the craft or the trail while the loop is still open then you loose a life. Tit for tat.

There are 'Chasers' at large in the research centre too. These take a particular dislike to robot craft and follow it around, along the ion trails, attempting to destroy it with a single touch. Every time your craft fills in a bit of the screen you gain points. A constantly decreasing bonus encourages you to be quick.

Once you have mastered the first screen with a single lepton, there are more in store with multiple leptons and plural chasers. The game is not graphically stunning, but then it doesn't have to be. It relies for its attraction, on your perseverance and its own ability to frustrate and egg you on for just one more go.

Name : Bumble Bee
Supplier : Micro Power
Price : £6.95
Reviewer : Geoff Bains
Rating : **

Yet another nail in Pacman's coffin, Bumble Bee is very similar in nature to Mr.P with a few embellishments. It is a bee that you steer around the maze, this time, and it is pollen that you

have to eat. The ghosts have turned to spiders in this version too. Rather than just a maze, you steer around a network of swing gates. These block off paths to the spider, but the bee can push through them. The odd (presumably poisonous) toadstool also lies in wait for unsuspecting drones. There are no power pills in this game, nor even power pollen, you have to kill the spiders by luring them into the barrels of fire that litter the garden.

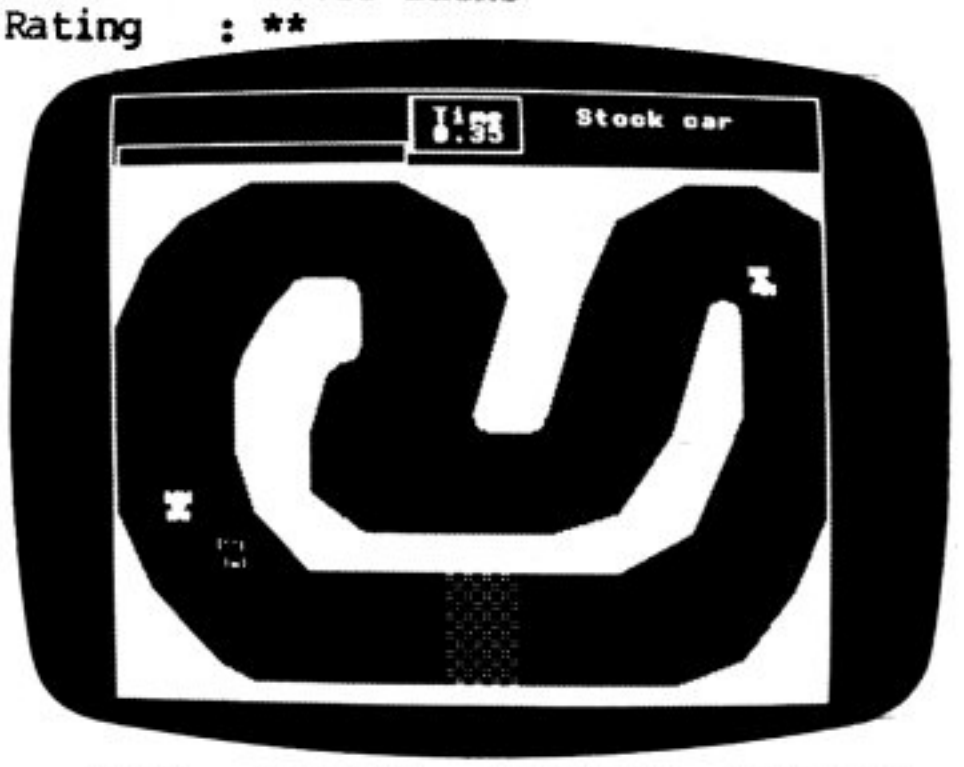
The theme of this game is, to say the least, contrived. However if it appeals to you, you won't be disappointed by the standard of programming. This is well up to Micro Power's usual quality. The graphics are good and the action smooth.

Name : Superfruit
Supplier : Simonsoft
Price : £5.95
Reviewer : Geoff Bains
Rating : **

Superfruit is a fruit machine simulation game. All the features that you'd find on the real thing are there for the trying. Holds, nudges, win-lines, gambles, and so on, along with features that you perhaps wouldn't expect; like a 'bounce' on the reels as they settle into position.

The only problem with Superfruit is that it is tediously slow. Gamble options that are supposed to flash before your eyes, limply meander across the screen. The reels move so slowly that you can easily fall asleep half way through. However, despite this lack of speed (far more the fault of the machine than the program) this game is an excellent simulation and a boon to the penniless addict.

Name : Stock Car
Supplier : Micro Power
Price : £6.95
Reviewer : Geoff Bains
Rating : **



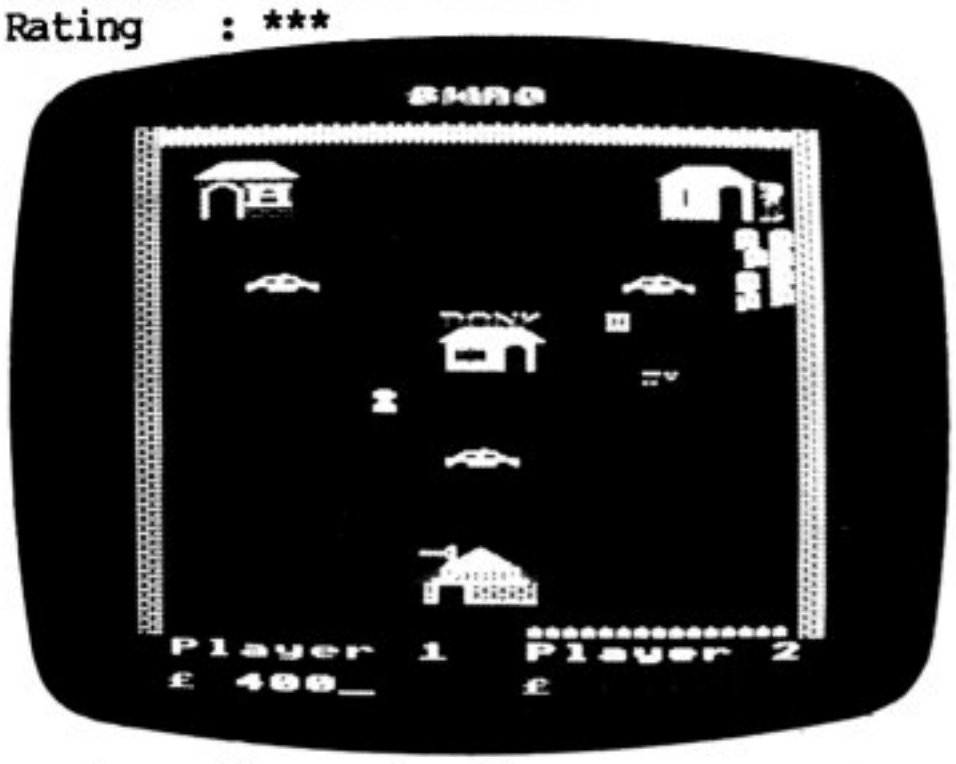
Rare amongst computer software, Stock Car is a two player game. If you are short of friends the Electron will happily take the opposing role.

You drive a car around one of six stock car racing circuits, displayed from above. The only controls are left and right steering, and up and down, through the four gears. If that's too simple for you, you'll be pleased to hear that there are various obstacles to contend with.

There are three opposing cars driving round the circuit. One of these is controlled by the other (human) player if that option is taken up. Oil slicks litter the circuits and the corners have the unusual feature of variable skid! Unlike the real thing, there is little chance of getting hurt in this game. If you hit an opposition car you don't crash, you don't lose a life, you just slow down a little. Punishment enough as speed is the winning factor in this game.

The stock car graphics are not realistic. The cars bend and distort as they corner and the whole game stops for a short break as the oil slicks change position. The idea of this game is not a new one and the implementation does not make the most of the Electron.

Name : $wag
Supplier : Micro Power
Price : £6.95
Reviewer : Alan R Webster
Rating : ***

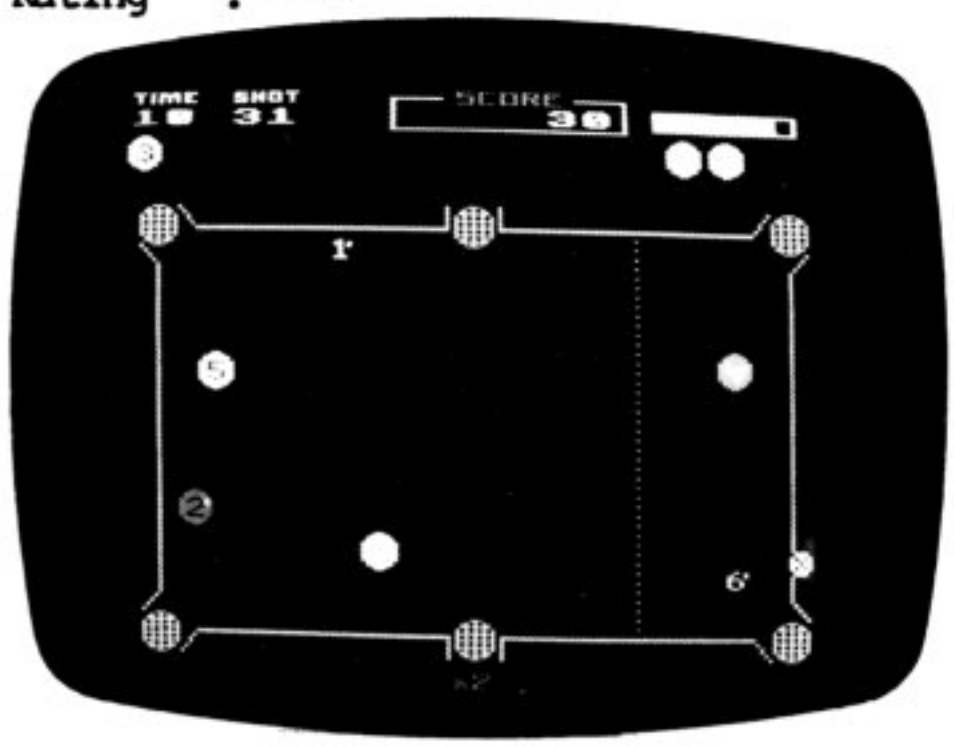


$wag is a brand new one or two player game for the Electron featuring mode 2 multi-coloured graphics and a practice feature where you play against the computer.

The game of $wag involves you pitting your wits against the other player to become the first person to steal £250,000 in diamonds. There are several killer droids that try to hamper you in your task of stealing the diamonds. You can also shoot at police cars, but this has the effect of making them angry, and they then give chase (giving extra money to your opponent). Apparently the only way you can stop these police cars from chasing you is by "...drinking a can of bear..."! (is this a sly advert for a famous lager?)

Overall $wag is a very hectic and enjoyable game, giving great value for money.

Name : Super Pool
Supplier : Software Invasion
Price : £7.95
Reviewer : Alan R Webster
Rating : ****



At last, an arcade game with no lasers! Super Pool is a simulation of Pool, the table game, but on a micro.

It's a scaled down version of the real game with just the one white cue ball, and six coloured balls numbered from 1 to 6.

The object of the game is first to pot all of the balls, and to do this you are allowed 60 seconds for each shot. Each time you pocket one of the balls, you gain a number of points and these points can be increased if you pot another ball in the same pocket.

The game features excellent graphics, and is both smooth and realistic. This is an excellent example of what can be achieved on an Electron.

HINTS HINTS HINTS HINTS HINTS HINTS HINTS HINTS HINTS HINTS

IF-THEN-ELSE CONSTRUCTS

If you are using IF <condition> <action>, and your action happens to be a * command, then Basic requires the THEN to be present, or it gets confused as to what exactly you mean.

Also, if you wish to perform a test for the opposite of a condition, for example the opposite of:

```
IF A=C AND B=D THEN ....
```

then, instead of:

```
IF NOT (A=C AND B=D) THEN ....
```

you can use

```
IF A=C AND B=D ELSE ....
```

RED ALERT

by Alan Barratt and David Green

If chess is too time consuming, solitaire too anti-social and draughts too predictable, then try this simple to learn yet challenging and decidedly unpredictable board game.

RED ALERT is a game of Scandinavian origin for two players. It is played on a six by six square board. Players take turns to place one counter at a time onto the board.

One player has pale blue counters and the other yellow. You are not allowed to place a counter directly on top of an opponent's counter but you can, and indeed should, build up on your own.

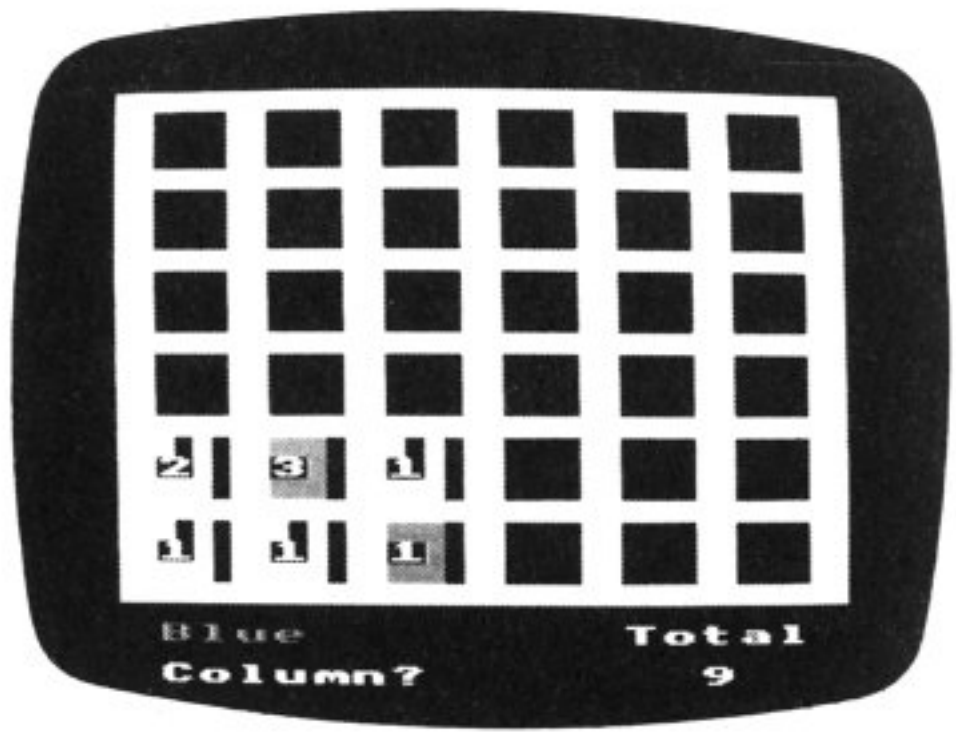
Each square has a critical mass. This value is the number of adjacent squares (horizontal and vertical) which the square has. This means that the critical masses are as follows:

Corner squares ... 2
Edge squares 3
Inner squares 4

Once the number of counters on any square reaches that square's critical mass the square 'explodes' and the counters disperse, one onto each adjacent horizontal and vertical square, leaving the original square now empty. This is the way to capture your opponent's counters: arrange an explosion next to an occupied square. Any counters on these adjacent squares become your own, and remain on these squares.

When well into a game, with several counters across the board, exploding one square can lead to a chain reaction that spreads right across the board. This gives Red Alert its unpredictability and challenge. It is quite possible to be down to two or three counters with your opponent having fifty, and then see him wiped out with a single, clever, move.

The winner is the player who eliminates all his opponent's counters. Alternatively, the game may be played for a fixed number of moves, the winner being the player with the most counters at that stage.

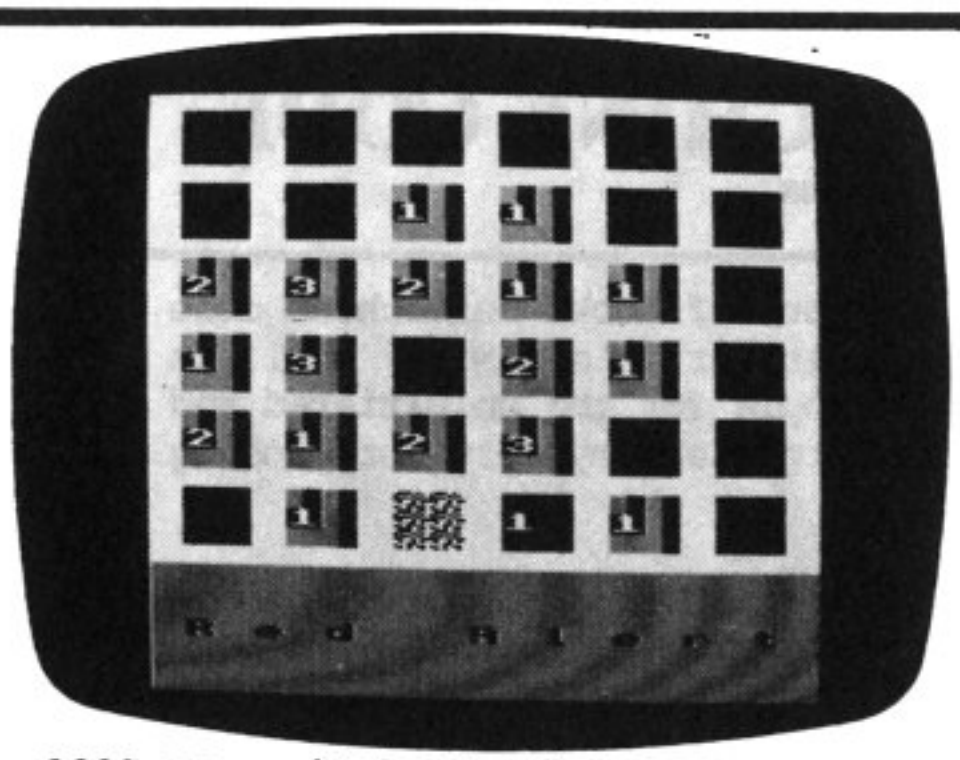


A counter is placed by designating the column and row positions. These both count from the bottom left hand corner of the board. The total number of counters on the board is displayed at the bottom of the screen at all times.

Red Alert makes colourful use of the Electron's mode 2 display. If you only have a black and white TV or monitor, the counters are shaped differently so that you can tell your own from your opponent's.

```
 10 REM Program RED ALERT
 20 REM Authors Alan Barratt
 30 REM         & David Green
 40 REM Version 1.0
 50 REM ELBUG   NOV 1984
 60 REM PROGRAM SUBJECT TO COPYRIGHT
 70 :
 80 ON ERROR GOTO 2680
 90 :
100 MODE 1
110 PROCinstr
120 MODE 2
```

```
 130 PROCsetup
 140 PROCscreen
 150 PROCgame
 160 PROCfinish
 170 IF Z<3 THEN RUN
 180 END
 190 :
1000 DEF PROCgame
1010 REPEAT
1020 Z=Z+1:Z1=3-Z:B=B+1:C2=0
1030 VDU19,4,4;0;
1040 COLOUR 128:CLS
1050 PRINT TAB(1,1) PLAYER$(Z MOD2)
1060 COLOUR 7
1070 PRINT TAB(13,1)"Total"TAB(15,3);B
-1
1080 FOR I=1 TO 2
1090 PRINTTAB(3*I-2,1+2*I);CO$(I);
1100 *FX15,0
1110 REPEAT
1120 XY$(I)=GET$
1130 UNTIL ASC(XY$(I))>48 AND ASC(XY$(
I))<55
1140 PRINT XY$(I)
1150 SOUND1,-15,220,4
1160 PROCwait(0.1)
1170 NEXT I
1180 X%=VAL(XY$(1)):Y%=VAL(XY$(2))
1190 IF new$(X%,Y%)=XO$(Z1) PROCsorry:
GOTO1050 ELSE new$(X%,Y%)=XO$(Z):new(X%
,Y%)=new(X%,Y%)+1
1200 REPEAT
1210 C1=0:C3=0:C4=0
1220 FOR X%=1 TO 6
1230 FOR Y%=1 TO 6
1240 IF NOT(new$(X%,Y%)=old$(X%,Y%) AN
D new(X%,Y%)=old(X%,Y%)) PROCplot(X%,Y%
,Z)
1250 IF C2<2 AND new(X%,Y%)>=D(X%,Y%)
PROCupdate
1260 IF new$(X%,Y%)=XO$(Z1) C3=1:C4=1
1270 NEXT Y%
1280 NEXT X%
1290 IF C4=0 C2=C2+1 ELSE IF C2=3 C1=0
1300 UNTIL C1=0:Z=2-Z
1310 UNTIL C3=0 AND B>1
1320 ENDPROC
1330 :
1340 DEF PROCsetup
1350 DIM new(6,6),old(6,6),new$(6,6),o
ld$(6,6),D(6,6)
1360 DIM XO$(2),OX$(2,3),PLAYER$(1),CO
$(2),XY$(2)
1370 PLAYER$(0)=CHR$17+CHR$6+"Blue"
1380 PLAYER$(1)=CHR$17+CHR$3+"Yellow"
1390 VDU24,0;224;1215;1023;
1400 VDU28,0,31,18,25
1410 VDU23,224,&FF,&FF,&FF,&FF,&FF,&FF
,&FF,&FF
1420 VDU23,225,&F0,&F0,&F0,&F0,&F0,&F0
,&F0,&F0
1430 VDU23,1,0;0;0;0;
1440 FOR X%=1 TO 2
1450 FOR Y%=1 TO 3
1460 READ A,B
1470 OX$(X%,Y%)=CHR$A+CHR$B
1480 NEXT Y%
1490 NEXT X%
1500 CO$(1)="Column?":CO$(2)="Row?"
1510 XO$(1)="X":XO$(2)="old"
1520 FOR X%=1 TO 6
1530 FOR Y%=1 TO 6
1540 READ D(X%,Y%)
1550 new$(X%,Y%)=" ":old$(X%,Y%)=" "
1560 NEXT Y%
1570 NEXT X%
1580 B=0:Z=0
1590 GCOL0,135:CLG
1600 PROCalert
1610 ENDPROC
1620 :
1630 DEF PROCwait(new)
1640 TIME=0
1650 REPEAT UNTIL TIME > 50*new
1660 ENDPROC
1670 :
1680 DEF PROCsorry
1690 SOUND1,-15,112,5
1700 SOUND1,-15,100,10
1710 SOUND1,-15,96,1
1720 SOUND1,-15,100,10
1730 COLOUR 12:PRINT TAB(11,3);"Not"TA
B(11,5)"Allowed"
1740 COLOUR 0:PROCwait(4):CLS
1750 ENDPROC
1760 :
1770 DEF PROCclear(X%,Y%)
1780 VDU5
1790 GCOL0,4
1800 FORE%=1TO3
1810 MOVE(X%*3-2)*64,252+(Y%*4-4+E%)*32
1820 PRINT CHR$224+CHR$224
1830 NEXT
1840 VDU4
1850 ENDPROC
1860 :
1870 DEF PROCscreen
1880 VDU19,7,11;0;19,4,11;0;
1890 FOR X%=1 TO 6
1900 FOR Y%=1 TO 6
1910 PROCclear(X%,Y%)
1920 NEXT Y%
1930 NEXT X%
1940 VDU19,4,4;0;19,7,7;0;
1950 ENDPROC
1960 :
1970 DEF PROCplot(X%,Y%,Z)
1980 PROCclear(X%,Y%)
```




```
 1990 IF new$(X%,Y%)<>" " VDU5:GCOL0,3*
Z:FOR E%=1 TO 3:MOVE(X%*3-2)*64,252+(Y%
*4-4+E%)*32:PRINT OX$(Z,4-E%):NEXT E%:M
OVE(X%*3-3)*64,252+(Y%*4-2)*32:GCOL0,7:
PRINT" ";new(X%,Y%):VDU4
 2000 old(X%,Y%)=new(X%,Y%)
 2010 old$(X%,Y%)=new$(X%,Y%)
 2020 C1=1
 2030 ENDPROC
 2040 :
 2050 DEF PROCupdate
 2060 VDU19,4,1;0;:PROCalert
 2070 IF X%<>6 new$(X%+1,Y%)=new$(X%,Y%
):new(X%+1,Y%)=new(X%+1,Y%)+1
 2080 IF X%<>1 new$(X%-1,Y%)=new$(X%,Y%
):new(X%-1,Y%)=new(X%-1,Y%)+1
 2090 IF Y%<>6 new$(X%,Y%+1)=new$(X%,Y%
):new(X%,Y%+1)=new(X%,Y%+1)+1
 2100 IF Y%<>1 new$(X%,Y%-1)=new$(X%,Y%
):new(X%,Y%-1)=new(X%,Y%-1)+1
 2110 new(X%,Y%)=new(X%,Y%)-D(X%,Y%)
 2120 old(X%,Y%)=10
 2130 IF new(X%,Y%)=0 new$(X%,Y%)=" "
 2140 PROCsmash(X%,Y%)
 2150 PROCplot(X%,Y%,Z)
 2160 ENDPROC
 2170 :
 2180 DEF PROCsmash(X%,Y%)
 2190 VDU5
 2200 PROCwait(1)
 2210 FOR Q%=5 TO 255 STEP 50
 2220 GCOL0,RND(15)
 2230 VDU23,240,RND(Q%),RND(Q%),RND(Q%)
,RND(Q%),RND(Q%),RND(Q%),RND(Q%),RND(Q%)
 2240 FORE%=1TO3
 2250 MOVE(X%*3-2)*64,252+(Y%*4-4+E%)*3
2:PRINTCHR$240+CHR$240
 2260 NEXT E%
 2270 NEXT Q%
 2280 VDU4
 2290 PROCclear(X%,Y%)
 2300 ENDPROC
 2310 :
 2320 DEF PROCalert
 2330 ENVELOPE1,1,4,-4,4,10,20,10,127,1
27,0,0,127,126
 2340 SOUND1,1,100,30
 2350 COLOUR 139:CLS:COLOUR 12
 2360 PRINTTAB(1,3);"R e d   A l e r t"
 2370 ENDPROC
 2380 :
 2390 DEF PROCfinish
 2400 COLOUR 132:CLS
 2410 SOUND1,-15,120,25
 2420 COLOUR12:PRINT TAB(3,1)"The Winne
r!"
 2430 COLOUR 0:PRINT TAB(4,3)"Score  ";
B;
 2440 COLOUR 7:PRINT TAB(4,5)"Press bar
";
 2450 *FX15,1
 2460 I=GET:CLS
 2470 PRINTTAB(2,2)"Another game ?";
 2480 REPEAT Z=INSTR("YyNn",GET$):UNTIL
 Z<>0
 2490 ENDPROC
 2500 :
 2510 DEF PROCinstr
 2520 COLOUR 129
 2530 PRINTTAB(10,2)SPC(15)
 2540 PRINTTAB(10,3)"  RED ALERT !  "
 2550 PRINTTAB(10,4)SPC(15)
 2560 COLOUR 128
 2570 PRINT TAB(3,8)"Capture your oppon
ent's squares by" ' "'exploding' your o
wn."
 2580 PRINT TAB(3,12)"The first player
to wipe out all" ' "trace of his oppone
nt wins the game."
 2590 PRINT TAB(3,15)"The critical mass
 of the squares" ' "varies around the b
oard:"
 2600 PRINT TAB(7,19)"Corner squares...
...2"
 2610 PRINT TAB(7,21)"Edge squares.....
...3"
 2620 PRINT TAB(7,23)"Inner squares....
...4"
 2630 COLOUR 1
 2640 PRINT TAB(6,30)"PRESS SPACE BAR T
O START"
 2650 REPEAT UNTIL GET=32
 2660 ENDPROC
 2670 :
 2680 ON ERROR OFF
 2690 MODE 6
 2700 IF ERR<>17 REPORT:PRINT" at line
";ERL
 2710 END
 2720 :
 2730 DATA 225,225,32,225,224,225,224,2
25,32,225,224,225
 2740 DATA 2,3,3,3,3,2,3,4,4,4,4,3,3,4,
4,4,4,3,3,4,4,4,4,3,3,4,4,4,4,3,2,3,3,3
,3,2
```

INTRODUCING MACHINE CODE

by Mike Williams

This month we have reviewed a number of books on machine code programming for the Electron. In this article, Mike Williams provides a short introduction to this topic for the uninitiated and helps to clear a path through all the jargon and mystery surrounding machine code.

Occasionally, in some of the programs that we have presented in previous issues of ELBUG, we have drawn your attention to short sections of so called 'machine code' included in those programs. Until now we have not attempted to explain in any detail what is meant by machine code. You may also have seen that there is a substantial chapter in your Electron User Guide devoted to something called 'assembly language', which you may also be wondering about.

In this article, we will attempt to explain exactly what these two phrases refer to, the reasons why machine code and assembly language are important to the Electron, and how this all fits in to the more familiar world of Basic.

BINARY BEGINNINGS

As you may know, all computers including the Electron, store all their information in memory in the form of 'binary' numbers. These are numbers to the base of two and are written with the digits '0' and '1' only. Thus two is represented as 10, five is 101, and a decimal ten is 1010. Like many micros the Electron uses the 6502 micro-processor chip, and this processor will recognise and act upon a set of comparatively simple instructions known just as 6502 code. It is these instructions which are called machine code, and programs using these instructions are called machine code programs. Of course, not all computers use the 6502 processor, and hence every different processor has its own unique brand of machine code.

NOW HEXADECIMAL

Now it would be very tedious for us to write machine code programs in binary, and so a mathematical shorthand called 'hexadecimal' is often used instead. This is based on 16 so that every four binary digits can be represented by a single hexadecimal digit. The complete set of hexadecimal digits is shown in the table below with their binary and decimal equivalents.

hex	binary	decimal	hex	binary	decimal
0	0000	00	8	1000	08
1	0001	01	9	1001	09
2	0010	02	A	1010	10
3	0011	03	B	1011	11
4	0100	04	C	1100	12
5	0101	05	D	1101	13
6	0110	06	E	1110	14
7	0111	07	F	1111	15

You will often find machine code represented in this form, and hexadecimal is often used for data as well at the machine code level.

ASSEMBLY LANGUAGE

Now although hexadecimal numbers are much less tedious to write than binary, any program written this way will be completely unreadable to all but the most fanatic of programmers. To make life easier for everyone, all 6502 machine code instructions can be represented by three letter mnemonic codes, that is codes which help us to remember their function. For example, you will find 'LDA' meaning 'load the accumulator with a number', or and 'STA' meaning 'store the number from the accumulator in memory'. This form of machine code also allows us to use names to refer to memory locations, rather than having to remember all the time the exact memory addresses. This form of machine code is called an 'assembly language', and although it may look very different, is functionally equivalent to machine code described earlier. Assembly language versions of machine code are just that much easier for us to write and understand.

THE ELECTRON'S ASSEMBLER

Unfortunately, the 6502 processor cannot understand assembler (another term for assembly language) and so a special program, called an 'assembler' is required to convert the mnemonic and symbolic codes into binary. On some machines this is a separate utility, but on the Electron the assembler is built into Basic, so that it is always immediately available whenever it is required. You just have to know how to call it. This is very easy. In any BBC Basic program the character '[' indicates that everything to follow is in assembler until the corresponding character ']' switches back to Basic. However, there is much more to writing machine code programs on your Electron than just using square brackets (hence the books on this subject reviewed this month).

To summarize, machine code and its representation called assembler, is the set of instructions directly understood and acted upon by the 6502 processor. Assembly language routines can be easily embedded in Basic programs and converted (or assembled) into proper machine code by Basic's built-in assembler.

BUT WHY USE MACHINE CODE?

Since BBC Basic is such a good language, why do we need assembler anyway. There are two principal answers to this question. Firstly, machine code programs will often take up much less memory than their Basic equivalents, and this is very useful on the Electron, especially if one of the 20K graphics modes is to be used with a large program.

Secondly, and most importantly, programs written directly in machine code (or assembler) will run much faster than if written in Basic. Machine code programs are often as much as 10 times faster, and sometimes the improvement can be greater still. This is partly because machine code allows you to control much more directly what is going on inside the computer, and also because the Basic interpreter, which is what enables your program in Basic to be carried out by the computer, takes up a lot of the processor's time, substantially slowing down the speed of your own program. That's why most of the commercial programs you can buy for your Electron, particularly action games, are written in machine code.

So, if your enthusiasm is now fired to learn all about machine code and write that best selling arcade game read our review in this issue of some of the books on this subject that you can go out and buy.

HINTS HINTS HINTS HINTS HINTS HINTS HINTS HINTS

PLUS-1 USER PORT - P. Wells

If you own a Plus-1 then you might be interested to know that you can use the printer port to perform the output functions of a user port (as on the BBC micro). The address of the port is &FC71. For example: to make the pattern, 00001111 (&0F in hex), appear on the eight output lines you would use ?&FC71=&0F.

ERROR IN THE USER GUIDE

There is an error in the Electron User Guide under the section on page 236, which explains about using 'negative INKEYs' from assembler. The correct way to call the routine for a 'negative' call is for the Y register to hold &FF, and the X register to hold the negative INKEY value.

DELAY LOOP

A REPEAT UNTIL TIME ... delay loop doesn't have to use positive values, as can be demonstrated by:

```
TIME=-500
REPEAT UNTIL TIME>0
```

This also means that different delay times (maybe set in another program) can use the same delay loop instruction.

BOOKS FOR MACHINE CODE PROGRAMMERS

Reviewed by Mike Williams

Many micro users have the urge to get to grips with machine code programming. Maybe they have heard of the power of machine code as evidenced by the best of the action games you can buy in the shops, maybe it's the desire to learn something different, to be one up on your friends. Whatever the reason there are many books out there to help you and this month we look at four of the most popular that deal specifically with machine code programming on the Electron.

As you know, the Electron is based very strongly on the BBC micro, and it will come as no surprise therefore to learn that all four of these books were originally written for that machine and have now been rewritten for the Electron. This at least has the advantage that there has been time for the programs to have been tested and any errors found and corrected before the Electron version appears in the shops. On the other hand there is an obvious temptation to the author or publisher to skimp the job of conversion.

If you are not too sure what machine code is anyway, or why you might want to program in another language then read the introduction to machine code in this issue designed to answer just such questions. Now let's see how these four books shape up.

Assembly Language Programming for the Acorn Electron by Ian Birnbaum, published by Macmillan at £7.95.

The initial impression is of a book that is well produced and is attractive in appearance. The text and program listings are very readable and the diagrams clear and well presented.

This book adopts a distinctive approach to the task of introducing machine code programming. It assumes and builds on a reasonable knowledge of Basic using short example programs to illustrate the points to be made. This is used extensively at the beginning of the book, but as the text develops, there is less and less reference to Basic and new machine code ideas are introduced directly. The use of Basic in this way is one which is likely to appeal to many Electron owners and it has certainly proved successful in the BBC version of this book.

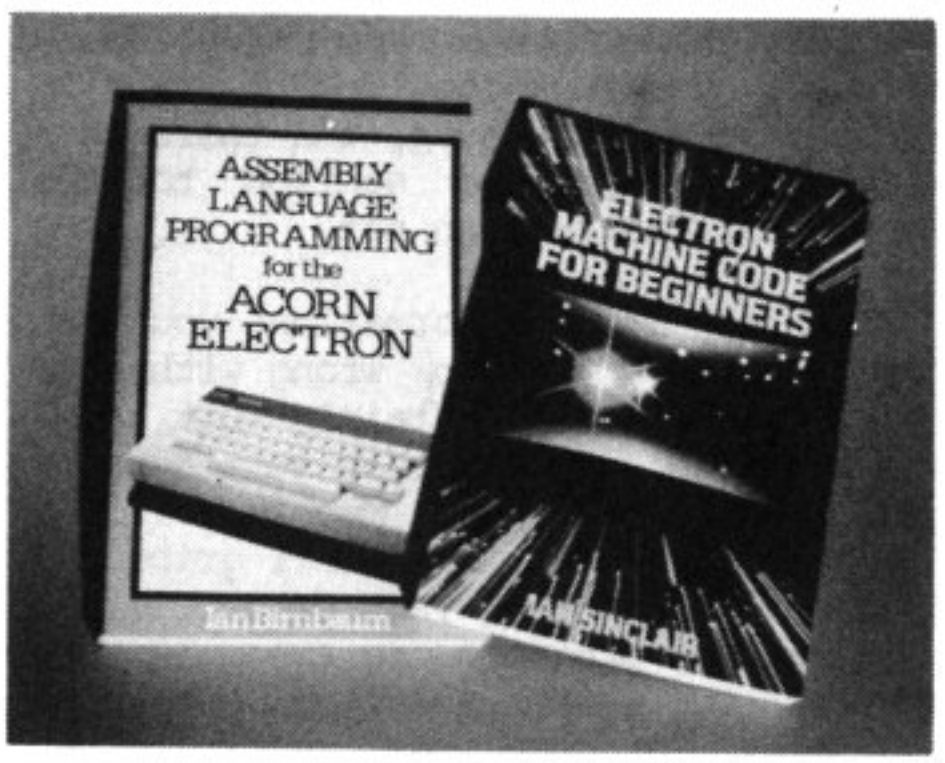
The book covers quite comprehensively all the main aspects of assembly language programming including decision making, looping, indexed and indirect addressing, arithmetic, subroutines and interrupts. Indeed the chapters on indexed and indirect addressing are models of excellence. The book shows clearly how to use assembler on the Electron and there are many excellent examples throughout the text.

Each chapter contains several exercises on the appropriate topic and solutions to all the exercises are contained in an appendix. There are several other appendices of useful reference information (including a complete description of the 6502 instruction set) while the last chapter in the book contains several complete utility programs.

With just over 300 pages in all, this is without doubt a book to be recommended and I am sure that the Electron edition will prove as popular as the earlier version for the BBC micro.

Electron Machine Code for Beginners by Ian Sinclair and published by Granada at £6.95.

This is in the nature of a more introductory text than the previous book. The first three chapters are devoted to a general discussion of the internal workings of the Electron including topics such as binary and hexadecimal numbers. Following this



grounding, the author introduces the basic features of the 6502 processor and its registers. The intricacies of both indexed and indirect addressing are introduced quite quickly together with all the branch instructions, and all this before any examples of actual machine code programs. I think that the relative paucity of example programs plus their delayed appearance (not until after page 60 does the first appear) would be my main criticism of this book.

Further chapters deal with input and output using the routines built into the operating system, and the debugging of machine code programs using a monitor program (The book refers extensively here to EXMON, a machine code monitor produced by BEEBUG, the Electron version of which is now available). A final 'Round-up' chapter and five appendices complete the 150 plus pages of this well produced book.

This book provides a most readable introduction to the whole world of machine code programming, but it is only an introduction and I suspect that most people would find the need for a further book to become reasonably proficient. In fairness, the author makes this point himself in the preface.

Assembly Language Programming on the Electron by John Ferguson & Tony Shaw, published by Addison Wesley at £7.95.

I do not feel quite as enthusiastic about this book as with the others reviewed here. It is not that it is in any way particularly bad but there are a number of small points about this book that overall created my less favourable impression. The book contains many diagrams, in itself a feature to be applauded, but in many cases these have been embellished in ways that are intended to add some humour to what is otherwise a rather serious subject, but which I often found irritating, confusing and irrelevant. The paper used is so thin that the printing on one side is often visible from the other, which does not help. The printed listings of the many example programs are often really too feint - the other books all show this is avoidable even if the listings are taken directly from a dot matrix printer.

This book provides a straight-forward introduction to many aspects of machine code programming and as mentioned above, there are plenty of example programs included from quite early in the book. I do find the very early use of the stack and subroutines rather surprising, but perhaps I am just a traditionalist in this respect. Overall, the coverage of assembly language programming is quite comprehensive (including interrupts for example), and you should be able to learn much from the many examples.

With just under 200 pages this book is good value but you will have to judge for yourself whether the style of presentation is one which appeals to you or not.

Electron Assembly Language by Bruce Smith, published by Shiva at £7.95.

This book is certainly packed with masses of information for the potential machine code programmer, and in fact provides a most useful reference guide for the more experienced user. Many of the chapters are quite short so that in some cases I would have wished for more examples of programs using the new information presented.

The early chapters introduce useful background ideas, including both binary and hexadecimal numbers, logical operations, memory layout, registers

etc. The next few chapters deal with the essence of machine code programming (addressing modes, branch instructions, the stack, subroutines and jumps). There are also several chapters dealing with various topics more specific to the Electron, including the interaction between Basic and machine code (particularly with CALL and USR), and the use of the operating system routines which are readily accessible in machine code.

The book also contains over 70 pages (out of a total of 200) of appendices. One of these shows how many Basic keywords and functions may be easily written in machine code, and there is a very full reference section to the whole 6502 instruction set.

With so much information you are hardly likely to go wrong with this book which should certainly suit those who wish to do more than just scratch the surface of machine code programming. I have one small gripe and that concerns the lack of sufficient application programs in assembler, as opposed to just short illustrative routines. But then you can't expect everything.

CONCLUSIONS

As I have indicated, all of these books will provide you with good value, and any choice is bound to be a matter of personal taste and preference. If you want a good introduction then go for those by Sinclair or Birnbaum, if you want comprehensive coverage and more detail, then those by Birnbaum and Smith are my choice, and if I really had to pick just one then it would have to be Ian Birnbaum's, which for me is 300 pages of sheer delight!

HINTS HINTS HINTS HINTS HINTS HINTS HINTS HINTS HINTS

COLOURFUL GCOL PARAMETERS - Paul Watts

As you may know, you can get the operating system to draw in two colour stripes by using the GCOL statement with silly parameters. Here is a routine that allows you to specify the two colours yourself. The displayed colours will, of course, be dependent upon the current chosen mode.

```
DEF PROCgcol(option,first,second) LOCAL a,b
GCOL option,first:a=?&359 AND &AA:GCOL option,second
b=?&359 AND &55:?&359=a + b:ENDPROC
```

INVERSE VIDEO - Ashley Denninson

To produce reverse video without resorting to a sequence of COLOUR statements, use ?&D3=255 to produce inverse video, and ?&D3=0 to revert back to normal video. (Note that modes with more than two colours will need different values - these can be found by experimenting.)

NUMBER OF CHARACTERS PER LINE

If you wish to use the full width of the screen, dependent on the mode in use, then the following formula will enable you to calculate this, given the current mode (C% will hold the number of characters on exit, and M% holds the mode number on entry):

```
IF M%<6 C%=2^(2-(M% MOD 3))*20 ELSE C%=40
```

WEE SHUGGY

by Hugh Darby

If you enjoyed Block Blitz published in ELBUG Vol.1 No.4 (acclaimed by many members as one of our best ever games) then you will certainly like Wee Shuggy, a game about a small guy trapped in a dungeon. He needs your help to find several keys and escape from the nightmare world in which he finds himself.

Wee Shuggy is a version of the popular computer game, 'Manic Miner', in which the player has to fetch all of the keys to open a grating so that he can escape from the dungeon. There are various obstacles on the way including thorn bushes, floors that collapse as you walk on them, stalactites and conveyor belts.

You lose a life if you jump into a stalactite or a thorn bush, and then have to start the current screen from the beginning again.

You have three lives in all, and there is a high score table and instructions in the program. The game uses just three keys to control the movements of the man, 'Z' and 'X' for left and right, and 'Return' for jump.

There are six different dungeons in this fast and colourful game, and each one provides a challenging, enjoyable and often frustrating experience.

We are sure that once you start playing Wee Shuggy you will agree that this is one of the most outstanding games that we have published in ELBUG. The time and effort spent typing the program into your micro will be well rewarded.

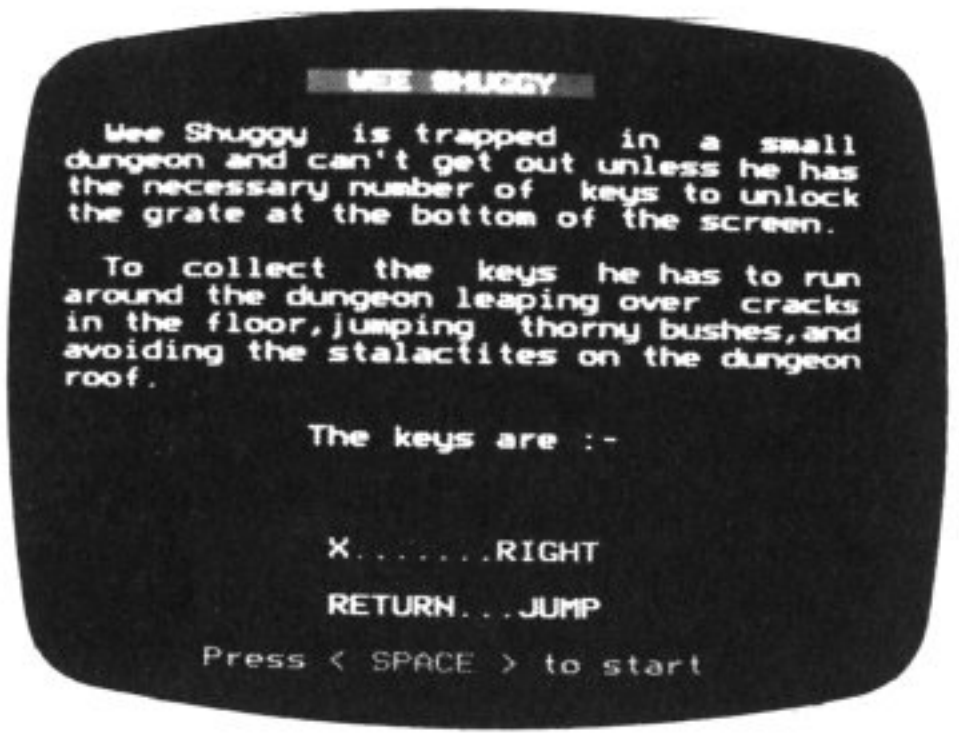
PROGRAM NOTES

The program is well structured, but care is still required when typing it in. Extensive use is made of user-defined characters set up in the procedure PROCcharacter from line 3490 onwards. These characters are used in displaying the different screens in the game (procedures screen, screen2, screen3, screen4, screen5, and screen6) by putting the character codes in a series of VDU statements in each case. The VDU statement is more concise as only the character code is required (224, 225 etc) rather than the character itself (CHR$224, CHR$225 etc) as would be needed in a PRINT statement. It is also easy to include any of the other VDU codes (in the range 0 - 31) to change colour, move the cursor etc as required.

For speed, the program also incorporates a short machine code routine PROCassem (to check on the screen character at the current cursor position) and uses direct memory access (the procedure PROCobject for example).

All the procedures have reasonably meaningful names so you should be able to identify their functions without too much difficulty.

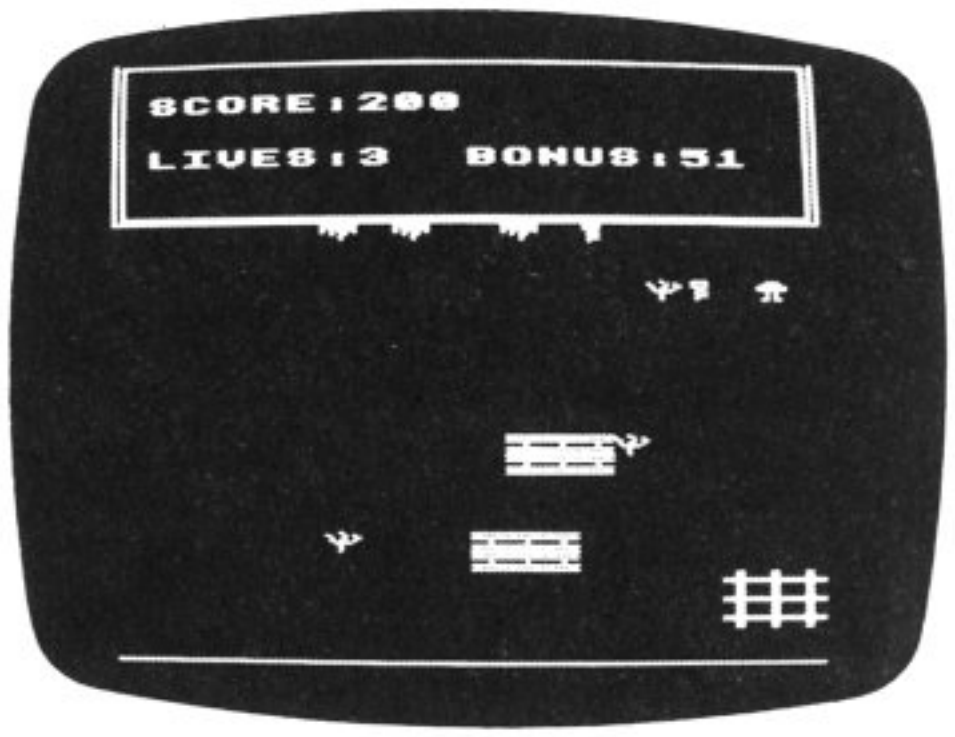
```
  10 REM PROGRAM WEE SHUGGY
  20 REM VERSION E0.1
  30 REM AUTHOR  HUGH DARBY
  40 REM ELBUG   NOVEMBER 1984
  50 REM PROGRAM SUBJECT TO COPYRIGHT
  60 :
 100 ON ERROR GOTO 3710
 110 MODE5
 120 PROCcharacter:PROCassem
 130 PROCinstr:PROCarray
 140 REPEAT
 150 MODE5:PROCvar
 160 PROChi:VDU23,1,0;0;0;0;
 170 REPEAT:CLS
 180 ONSCR%GOTO190,200,210,220,230,240
,250
 190 PROCscreen:GOTO260
 200 PROCscreen2:GOTO260
 210 PROCscreen3:GOTO260
 220 PROCscreen4:GOTO260
 230 PROCscreen5:GOTO260
 240 PROCscreen6:GOTO260
 250 PROCfinale
 260 PROCheader
 270 REPEAT
 280 PROCplayer:PROCtest
 290 UNTILdeadORfinish
 300 IFdeadPROCfinishELSEPROCnewscr
 310 UNTILdead
 320 MODE6:PROCend
 330 UNTILFALSE
 340 END
 350 :
1000 DEFPROCarray
1010 DIMN$(5),HI%(5):FORI%=1TO5:N$(I%)
="ELBUG MAG":HI%(I%)=3000-I%*500:NEXT
1020 ENDPROC
1030 :
1040 DEFPROCassem
1050 P%=&80
1060 [OPT2:LDA#135:JSR&FFF4:TXA:CLC:AD
C#96:TAX:STX&70:RTS:]
1070 ENDPROC
1080 :
1090 DEFPROChi
1100 VDU22,5:PRINT
1110 COLOUR3:PRINT" TODAYS TOP SHUGGYS"
1120 COLOUR2:FORI%=1TO5
1130 VDU31,0,I%*3+1
1140 PRINTSTR$(I%)+"..";TAB(3)N$(I%);T
AB(13)".."+STR$(HI%(I%))
1150 NEXT
1160 COLOUR1:PRINT''''"  Press <SPACE>"
1170 REPEATUNTILGET=32
1180 CLS
1190 ENDPROC
1200 :
1210 DEFPROCdouble(A$)
1220 A$=CHR$141+A$
1230 PROCcent(A$):PROCcent(A$)
1240 ENDPROC
1250 :
1260 DEFPROCcent(A$)
1270 pos=20-LENA$/2
1280 VDU31,pos,VPOS:PRINTA$
1290 ENDPROC
1300 :
1310 DEFFNspc=STRING$((10-LENN$(I%)),C
HR$32)
1320 :
1330 DEFPROCscreen
1340 CLS:COLOUR1:PRINTTAB(0,30);STRING
$(20,CHR$224);CHR$30;CHR$11
1350 VDU31,4,27,224,224,224,224,224,22
4,17,2,232,232,232,31,10,26,232,232,232
,17,1,224,225,225,225,225,224,224
1360 VDU31,0,22,224,224,224,32,32,32,2
24,224,224,224,224,17,2,232,232,232,17,
1,228,228,228,32,224,224,31,11,21,17,2,
232,232,232,17,3,229
1370 VDU31,0,18,17,1,224,224,31,0,14,2
24,224,224,224,224,224,224,224,225,225,
225,224,225,225,225,224,224,224,224,224
1380 VDU31,6,26,17,3,229,31,15,13,229,
17,2,231,31,19,11,231,31,4,10,231,32,17
,3,230,32,230,32,32,230,32,17,2,231
1390 KEYS=4
1400 ENDPROC
1410 :
1420 DEFPROCheader
1430 VDU19,3,2;0;
1440 IFG%<>0GOTO1460
1450 COLOUR2:FORI%=28TO30:VDU31,17,I%,
233,233,233:NEXT
1460 COLOUR132:FORI%=1TO9:PRINTTAB(0,I
%)SPC20:NEXT
1470 PRINTTAB(1,3)"SCORE:";SC%;TAB(1,6
)"LIVES:";liv%;TAB(10,6)"BONUS:60"
1480 MOVE0,704:DRAW0,992:DRAW1279,992:
DRAW1279,704:DRAW0,704
1490 MOVE16,712:DRAW16,984:DRAW1262,98
4:DRAW1262,712:DRAW16,712
1500 TIME=0
1510 ENDPROC
1520 :
1530 DEFPROCinstr
1540 VDU22,6
1550 PROCcent("WEE SHUGGY")
1560 PRINT'"  Wee Shuggy  is trapped
 in  a  small dungeon and can't get out
 unless he has the necessary number of
 keys to unlock the grate at the bottom
 of the screen."
 1570 PRINT'"  To  collect  the  keys
he has to run around the dungeon leapin
g over  cracks in the floor,jumping  th
orny bushes,and avoiding the stalactite
s on the dungeon roof."
```




```
1580 PRINT:PROCcent(" The keys are :-")
1590 PRINT:PROCcent("Z.....LEFT")
1600 PRINT:PROCcent("X.....RIGHT")
1610 PRINT:PROCcent("RETURN...JUMP")
1620 PRINT:PROCcent("Press < SPACE > t
o start")
1630 REPEATUNTILGET=32:CLS
1640 ENDPROC
1650 :
1660 DEFPROCvar
1670 X%=0:Y%=30:J%=0:DIR%=0:G%=0
1680 OX%=X%:OY%=Y%:SC%=0:liv%=3
1690 dead=0:finish=0:SCR%=1:keys=0
1700 ENDPROC
1710 :
1720 DEFPROCprint(c)
1730 IFc<>0GOTO1760
1740 *FX19
1750 VDU31,X%,Y%,32:ENDPROC
1760 *FX19
1770 VDU17,c,31,X%,Y%,253+X%MOD2
1780 ENDPROC
1790 :
1800 DEFPROCplayer
1810 PROCprint(0)
1820 IFFNcheck(X%,Y%+1)=0ANDJ%=0:Y%=Y%+
1:IFADVAL-6=15SOUND1,2,30-Y%,1:GOTO1900
1830 IFFNcheck(X%,Y%+1)<>0PROCobj2
1840 OX%=X%:OY%=Y%
1850 IFINKEY-74ANDJ%=0DIR%=INKEY-98-IN
KEY-67:J%=1:UNO%=-1:IFADVAL-6=15SOUND1,
3,1,2
1860 IFJ%<>0PROCjump:GOTO1890
1870 IFINKEY-67ANDX%<19X%=X%+1
1880 IFINKEY-98ANDX%>0X%=X%-1
1890 IFFNcheck(X%,Y%)<>0PROCobject
1900 COLOUR2:PRINTTAB(16,6);60-TIME DI
V 100;CHR$32
1910 PROCprint(2)
1920 ENDPROC
1930 :
1940 DEFFNcheck(x%,y%)
1950 VDU31,x%,y%:CALL&80
1960 IF?&70=96OR?&70=128THEN=0ELSE=1
1970 :
1980 DEFPROCobject
1990 IF?&70>=224AND?&70<229J%=0:UNO%=0
:Y%=Y%-1:ENDPROC
2000 IF?&70=229OR?&70=230OR?&70=234dea
d=-1:ENDPROC
2010 IF?&70=231PROCbonus:ENDPROC
2020 IF?&70=232X%=OX%:Y%=OY%:ENDPROC
2030 IF?&70=233ANDKEYS=keys finish=-1E
LSEIF?&70=233X%=16:Y%=30:ENDPROC
2040 IF?&70=235PROCprint(0):X%=6:Y%=19
2050 IF?&70=236PROCprint(0):X%=14:Y%=19
2060 ENDPROC
2070 :
2080 DEFPROCjump
2090 X%=X%+DIR%:J%=J%+1:Y%=Y%+UNO%
2100 IFFNcheck(X%,Y%)<>0PROCobject
2110 IFJ%>3UNO%=1
2120 IFJ%=7UNO%=0:J%=0
2130 ENDPROC
2140 :
2150 DEFPROCtest
2160 IFY%>30dead=-1:Y%=30ELSEIFY%<11PR
OCprint(0):Y%=11
2170 IFX%>18PROCprint(0):X%=18ELSEIFX%
<1PROCprint(0):X%=1
2180 IFTIME>=6000 dead=-1
2190 ENDPROC
2200 :
2210 DEFPROCcollapse
2220 VDU17,1
2230 IF?&70=227VDU31,X%,Y%+1,32:ENDPROC
2240 VDU31,X%,Y%+1,?&70+1
2250 ENDPROC
2260 :
2270 DEFPROCbonus
2280 SC%=SC%+100:keys=keys+1:VDU17,0,3
1,X%,Y%,32
2290 SOUND1,1,200,1
2300 COLOUR2:PRINTTAB(7,3);SC%
2310 ENDPROC
2320 :
2330 DEFPROCobj2
2340 IF?&70>224AND?&70<228PROCcollapse
:ENDPROC
2350 IF?&70=228X%=X%-1:ENDPROC
2360 IF?&70=233ANDKEYS=keys finish=-1E
LSEIF?&70=233X%=16:Y%=30:ENDPROC
2370 IF?&70=234dead=-1:ENDPROC
2380 IF(?&70=229OR?&70=230)ANDJ%=0Y%=Y
%+1:ENDPROC
2390 IF?&70=231ANDJ%=0Y%=Y%+1
2400 ENDPROC
2410 :
2420 DEFPROCnewscr
2430 T=TIME
2440 CLS:SCR%=SCR%+1:SC%=SC%+60-TDIV100
2450 X%=0:Y%=30:J%=0:DIR%=0:OX%=X%:OY%
=Y%:dead=0:finish=0:keys=0
```

```
 2460 IFSCR%<7 PRINTTAB(0,10)"GOING ONT
O SCREEN ";SCR%ELSEENDPROC
 2470 T=TIME:REPEAT UNTIL TIME=T+300
 2480 ENDPROC
 2490 :
 2500 DEFPROCfinish
 2510 CLS:FORI%=100TO0STEP-4:SOUND1,3,I
%,2:NEXT
 2520 liv%=liv%-1:IFliv%<=0dead=TRUE:EN
DPROC
 2530 X%=0:Y%=30:J%=0:DIR%=0:OX%=X%:OY%
=Y%:dead=0:finish=0:keys=0
 2540 IFTIME>=6000PRINTTAB(0,10)"YOU RA
N OUT OF TIME"
 2550 ENDPROC
 2560 :
 2570 DEFPROCend
 2580 *FX21,0
 2590 FORT%=255TO0STEP-1:SOUND&11,4,T%,
1:SOUND&10,4,7,1:NEXT
 2600 T=TIME:REPEAT UNTIL TIME=T+300
 2610 IFSC%<=HI%(5)ENDPROC
 2620 FORI%=5TO1STEP-1
 2630 IFHI%(I%)<SC%C%=I%
 2640 NEXT
 2650 PRINT'''':PROCcent("WELL DONE!")
 2660 PRINT':PROCcent("YOUR SCORE OF "+
STR$(SC%))
 2670 PRINT':PROCcent("IS ENOUGH TO RAN
K YOU "+STR$(C%)+MID$("stndrdthth",C%*2
-1,2))
 2680 PRINT':PROCcent("PLEASE ENTER YOU
R NAME ")
 2690 INPUTTAB(5,15)A$
 2700 IFLENA$>10CLS:GOTO2650
 2710 IFC%=5GOTO2740
 2720 FORI%=4TOC%STEP-1
 2730 HI%(I%+1)=HI%(I%):N$(I%+1)=N$(I%)
:NEXT
 2740 HI%(C%)=SC%:N$(C%)=A$
 2750 ENDPROC
 2760 :
 2770 DEFPROCscreen2
 2780 *FX21
 2790 VDU31,0,30,17,3:FORI%=0TO19:VDU23
4:NEXT:VDU30,11
 2800 VDU31,0,10,17,3:FORI%=0TO19:VDU23
0:NEXT
 2810 VDU31,0,15,17,1:FORI%=0TO19:VDU22
5:NEXT
 2820 VDU31,0,31,224,224,224,11,11,32,3
2,224,224,11,11,32,32,224,224,11,11,32,
32,224,224,31,8,30,228,228,228,228,32,3
2,10,225,225,224
 2830 VDU31,7,22,225,225,225,32,32,32,3
2,32,32,225,225,225,31,0,26,224,224,31,
3,22,224,224,31,5,18,224,224
 2840 VDU31,8,29,17,2,231,31,8,21,231,3
1,17,21,231,31,15,14,231,231,231
 2850 keys=0:KEYS=6
 2860 ENDPROC
 2870 :
 2880 DEFPROCscreen3
 2890 *FX21
 2900 VDU31,0,30,17,3:FORI%=0TO19:VDU23
4:NEXT:VDU30,11
 2910 VDU31,0,10,17,3:FORI%=0TO19:VDU23
0:NEXT
 2920 VDU31,0,31,17,1,224,224,225,31,3,
28,228,228,11,11,11,228,228,11,11,11,22
8,228,11,11,11,228,228,225,225,32,32,22
4,224,225,225,225
 2930 VDU31,0,27,224,224,31,0,23,224,22
4,31,0,19,224,224,31,17,27,224,224,31,6
,13,225,225,225,225,224,225
 2940 VDU31,3,16,225,225,225,31,11,27,2
24,224,31,15,20,17,2,232,8,10,232,8,10,
232,8,10,232,31,13,31,232,232,232,232
 2950 VDU17,2,31,0,26,231,31,18,26,231,
31,10,12,231,31,18,22,231,31,6,24,231
 2960 keys=0:KEYS=5
 2970 ENDPROC
 2980 :
 2990 DEFPROCscreen4
 3000 *FX21
 3010 VDU31,0,30,17,3:FORI%=0TO19:VDU23
4:NEXT:VDU30,11
 3020 VDU31,10,12,17,2:FORI%=0TO18:VDU2
32,10,8:NEXT:VDU232
 3030 VDU17,1,31,0,31,224,224,31,0,27,2
24,224,31,0,23,224,224,32,225,225,32,11
,11,224,11,11,224,11,11,224,31,0,14,228
,228,228,228,228,224,224,224
 3040 VDU31,11,12,225,225,225,31,11,13,
17,2,230,230,230,17,1,31,14,15,228,228,
31,12,18,228,228,31,11,31,224,224
 3050 VDU31,13,27,224,224,31,15,23,224,
224,31,17,19,224,224,17,2,31,16,27,232,
232,232,232,31,16,26,232,232,232,232,17
,1,31,16,31,224,31,18,15,224,224
 3060 VDU17,2,31,0,26,231,31,11,30,231,
31,19,14,231,31,19,25,231,31,0,13,231
 3070 keys=0:KEYS=5
 3080 ENDPROC
 3090 :
 3100 DEFPROCscreen5
 3110 *FX21
 3120 VDU31,0,30,17,3:FORI%=0TO19:VDU23
4:NEXT:VDU30,11
 3130 VDU31,10,10,17,2:FORI%=0TO20:VDU2
32,10,8:NEXT:VDU232
 3140 VDU31,16,13:FORI%=0TO17:VDU232,10
,8:NEXT:VDU232
 3150 VDU17,1:FORI%=13TO27STEP3:VDU31,1
7,I%,225,225,225:NEXT
 3160 VDU17,1,31,0,31,224,224,224,31,5,
31,224,224,31,8,31,224,224,31,8,27,224,
224,31,5,24,224,224,31,0,21,224,224,224
,31,0,17,224,224,224
```

```
 3170 VDU31,0,13,224,224,224,32,225,225
,225,225,225,31,9,20,224,11,32,8,11,32,
31,11,20,225,225,32,225,225,31,13,16,22
8,228,228,31,13,13,228,228,228,31,15,31
,225
 3180 VDU17,2,31,19,12,231,31,17,15,231
,31,19,18,231,31,17,21,231,31,19,24,231
,31,8,26,231,31,10,18,231,31,15,30,231
 3190 keys=0:KEYS=8
 3200 ENDPROC
 3210 :
 3220 DEFPROCscreen6
 3230 *FX21
 3240 VDU17,1,31,0,31,224,224,224,32,22
7,227,32,17,3,234,17,1,227,32,227,227,2
24,225,17,3,234,234,17,1,225,224,225,22
4,30,11,31,2,27,224,31,1,23,224,224,31,
1,19,235,31,5,20,228,228
 3250 VDU17,2:FORI%=31TO19STEP-1:VDU31,
3,I%,232,31,9,I%,232:NEXT
 3260 FORI%=27TO19STEP-1:VDU31,12,I%,23
2,232,31,16,I%,232:NEXT:VDU31,7,19,232,
8,10,232,8,10,232,31,6,29,232,8,10,232,
8,10,232
 3270 FORI%=0TO19:VDU31,I%,18,232,31,I%
,17,232:NEXT:VDU31,14,17,32,32,31,14,18
,32,32
 3280 VDU17,1,31,8,27,224,31,8,23,224,3
1,8,20,224,11,32,31,10,27,227,31,10,23,
227,17,3,31,10,24,230,31,11,21,230,17,2
,31,11,26,232,17,1,31,18,27,224,224,31,
18,23,224,224,31,18,19,236
 3290 VDU31,14,17,226,226,31,14,20,226,
226
 3300 VDU17,2,31,2,19,231,31,6,28,231,3
1,9,19,231,31,10,22,231,31,13,28,231,31
,16,28,231,31,17,16,231
 3310 FORI%=14TO16:VDU31,0,I%,233,233,2
33,231,231,231:NEXT
 3320 keys=0:KEYS=16:G%=1
 3330 ENDPROC
 3340 :
 3350 DEFPROCfinale
 3360 FORT%=0TO255:SOUND&11,4,T%,1:SOUN
D&10,4,7,1:NEXT:SOUND0,1,7,1
 3370 ENVELOPE1,1,0,0,0,0,0,0,0,0,-1,-1
,126,126
 3380 SOUND17,1,101,2:SOUND1,0,0,1:SOUN
D1,1,101,2:SOUND1,0,0,1:SOUND1,1,101,2:
SOUND1,0,0,1:SOUND1,1,125,5:SOUND1,0,0,
2:SOUND1,1,101,2:SOUND1,0,0,2:SOUND1,1,
125,5
 3390 COLOUR1:PRINTTAB(5,4)"WELL DONE!"
 3400 PRINTTAB(3,7)"WEE SHUGGY HAS";TAB
(6,9)"ESCAPED"
 3410 VDU31,0,21,17,2:FORI%=0TO19:VDU23
2,8,10,232,11:NEXT
 3420 Y%=20:X%=10:U=-1:REPEAT
 3430 IFY%>16ANDU Y%=Y%-1ELSEU=0:Y%=Y%+1
```




```
 3440 PROCprint(2):IFY%=20U=-1
 3450 T=TIME:REPEATUNTILTIME=T+3:PROCpr
int(0)
 3460 UNTIL0
 3470 END
 3480 :
 3490 DEFPROCcharacter
 3500 VDU23,224,255,170,255,170,255,170
,0,0
 3510 VDU23,225,255,129,255,129,255,129
,0,0
 3520 VDU23,226,0,0,255,129,255,129,0,0
 3530 VDU23,227,0,0,0,0,255,129,0,0
 3540 VDU23,228,146,146,255,129,255,255
,0,0
 3550 VDU23,229,16,138,73,82,60,16,16,16
 3560 VDU23,230,255,255,191,173,44,8,8,0
 3570 VDU23,231,60,36,60,16,28,16,28,16
 3580 VDU23,232,255,66,255,8,8,255,66,2
55
 3590 VDU23,233,24,24,24,255,255,24,24,
24
 3600 VDU23,234,0,0,0,146,73,36,146,73
 3610 VDU23,253,28,62,107,127,20,20,20,
54
 3620 VDU23,254,0,0,28,62,107,127,20,54
 3630 VDU23,235,255,254,252,248,240,224
,192,128,0
 3640 VDU23,236,255,127,63,31,15,7,3,1
 3650 ENVELOPE1,1,0,0,0,0,0,0,0,0,-2,-2
,126,0
 3660 ENVELOPE2,1,0,0,0,0,0,0,0,0,-5,-5
,95,0
 3670 ENVELOPE3,1,5,5,5,-5,-5,-5,15,15,
-9,-9,126,126
 3680 ENVELOPE4,0,0,0,-1,1,1,1,0,1,0,25
4,120,128
 3690 ENDPROC
 3700 :
 3710 ON ERROR OFF:MODE 6
 3720 IF ERR=17 END
 3730 REPORT:PRINT" at line ";ERL
 3740 END
```

Subscriptions and Software Help Line
St Albans (0727) 60263
Manned Mon-Fri 1pm-4pm.

BACK ISSUES AND SUBSCRIPTIONS

BACK ISSUES (Members only)

All back issues will be kept in print (from November 1983). Send 90p per issue PLUS an A5 SAE to the subscriptions address. Back copies of BEEBUG are available to ELBUG members at this same price. This offer is for members only, so it is ESSENTIAL to quote your membership number with your order. Please note that the advertising supplements are not supplied with back issues.

Subscription and Software Address

ELBUG
PO BOX 109
High Wycombe
Bucks

SUBSCRIPTIONS

Send all applications for membership, and subscription queries to the subscriptions address.

MEMBERSHIP COSTS:

U.K.
£5.90 for 6 months (5 issues)
£9.90 for 1 year (10 issues)
Eire and Europe
Membership £16 for one year.
Middle East £19
Americas and Africa £21
Elsewhere £23
Payments in Sterling preferred.

SOFTWARE (Members only)

This is available from the software address.

MAGAZINE CONTRIBUTIONS AND TECHNICAL QUERIES

Please send all contributions and technical queries to the editorial address opposite. All contributions published in the magazine will be paid for at the rate of £25 per page.

We will also pay £10 for the best Hint or Tip that we publish, and £5 to the next best. Please send all editorial material to the editorial address opposite. If you require a reply it is essential to quote your membership number and enclose an SAE.

Editorial Address

ELBUG
PO Box 50
St Albans
Herts

ELBUG MAGAZINE is produced by BEEBUG Publications Ltd.
Editor: Mike Williams.
Assistant Editor: Geoff Bains. Production Editor: Phyllida Vanstone.
Technical Assistants: David Fell and Alan Webster.
Managing Editor: Lee Calcraft.
Thanks are due to, Sheridan Williams, and Adrian Calcraft for assistance with this issue.

New Elbug Binders

We have produced an attractive hard-backed binder for the ELBUG magazine. These binders are green in colour with "ELBUG" in gold lettering on the spine and allow for the whole of one volume of the magazine to be stored as a single reference book.

Each binder will accommodate 10 ELBUG magazines, and is supplied with 12 wires to enable the index and the latest copy of the supplement to be included within the binder if required. Individual issues may be easily added and removed, allowing for the latest volume to be filed as it arrives.

The price of the new ELBUG binder is £3.90 including VAT, please add 50p post and packing for delivery within the U.K. Overseas members please send the same amount, this will cover the extra postage but not VAT.
Plese send to:
BEEBUGSOFT, PO BOX 109, High Wycombe, Bucks, HP10 8HQ.

THE BEST OF ELBUG ON CASSETTE

Many of the best programs published in ELBUG have been collected together and published by Penguin Books under the name "Games and other programs for the Acorn Electron" at £3.95. This book is part of the Penguin Acorn Computer Library and at present there is just one other title available though others are planned.

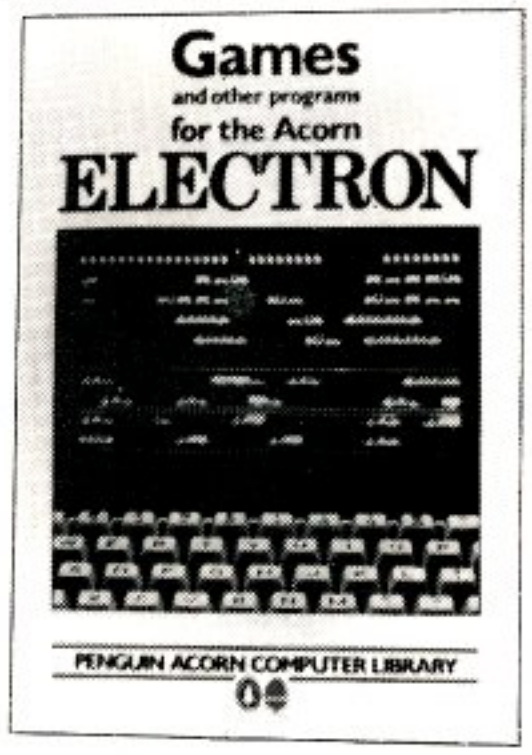
There are 20 programs in all in four different categories:

Action Games
- Munch-Man
- Mars Lander
- Invasion
- Robot Attack
- Hedgehog

Thought games
- Higher/Lower
- Five-Dice
- Life
- Anagrams
- Return of the Diamond

Visual Displays
- Union Jack
- Square Dance
- Ellipto
- Screenplay
- 3-D Rotation

Utilities
- Sound Wizard
- Bad Program Lister
- 3-D Lettering
- Bad Program Rescue
- Double Height Text



All 20 programs are now available on cassette from our software address (in High Wycombe) price £7 to members and £9 to non-members, plus 50p post & packing in either case.

ELBUG MAGAZINE CASSETTE

To save wear and tear on fingers and brain, we offer, each month, a cassette of the programs featured in the latest edition of ELBUG. The first program on each tape is a menu program, detailing the tape's contents, and allowing the selection of individual programs. The tapes are produced to a high technical standard by the process used for the BEEBUGSOFT range of titles.

Magazine cassettes have been produced for each issue of ELBUG from Volume 1 Number 1 onwards and are all available from stock, priced £3.00 each inclusive of VAT. See below for ordering information.

This months cassette includes:

Volume 2 Number 1

Wee Shuggy (a superb action packed arcade-style game), Whirlpool display, a universal joystick routine for use with Acorn's Plus 1, a program illustrating how to synchronise words and music, Downhill Ski Racer (another exciting action game), and a further fascinating and challenging game called Red Alert.

MAGAZINE CASSETTE SUBSCRIPTION

We are also able to offer ELBUG members subscription to the magazine cassette, this gives the added advantage of receiving the cassette at around the same time as the magazine each month. Subscriptions may either be for a period of 1 year or 6 months. (NOTE Magazine cassettes are produced 10 times each year).

If required, subscriptions may be backdated as far as Volume 1 Number 1, so when applying please write to the address below quoting your membership number and the issue from which you would like your subscription to start.

MAGAZINE CASSETTE ORDERING INFORMATION

Individual ELBUG Magazine Cassettes £3.00.
P & P: Please add 50p for the first and 30p for each subsequent cassette.
Overseas orders: Please send the same amount, this will include the extra post but not VAT.

Magazine Cassette Subscription

1 YEAR (10 issues)	£33.00 Inclusive.....	O'SEAS £39.00	No VAT payable
6 MONTHS (5 issues)	£17.00 Inclusive.....	O'SEAS £20.00	No VAT payable

Please be sure to specify that you require subscription to the ELBUG magazine cassette (as opposed to the BEEBUG cassette), and enclose your membership number with a cheque made payable to BEEBUGSOFT.

Please send to..

ELBUG Magazine Cassette, BEEBUGSOFT, PO Box 109, High Wycombe, HP10 8HQ

Printed in England by Staples Printers St Albans Limited at The Priory Press.